Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Auscultación Dinámica de Vías de Ferrocarril Usando Sensores de Aceleración y Posición

Autor: José María Rioboo León

Tutor: Antonio Luque Estepa

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías de Telecomunicación

Auscultación Dinámica de Vías de Ferrocarril Usando Sensores de Aceleración y Posición

Autor:
José María Rioboo León

Tutor:
Antonio Luque Estepa
Profesor titular

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Auscultación Dinámica de Vías de Ferrocarril Usando Sensores de Aceleración y Posición

Autor: José María Rioboo León

Tutor: Antonio Luque Estepa

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

Agradecimientos

A mi tutor, por la ayuda prestada y la paciencia y por aceptar y ayudarme en la idea que le propuse hace ya algún tiempo y que ahora vemos realizada en esta memoria.

A mi familia y amigos, por ser mi apoyo diario y los que me han ayudado a resolver todos los problemas, aportándome calma y sabiduría.

A mis padres, por ser el ejemplo que toda persona querría tener en su vida y por darme esos valores con los que me siento tan identificado y una educación que ha conseguido llevarme hasta aquí y ser quien soy hoy día.

Y a Paula, gracias por ser la mejor persona que alguien puede tener a su lado, por el apoyo incondicional, por creer en mí cuando hasta yo dejo de hacerlo y por querer compartir tu vida conmigo, si de algo estoy más orgulloso que de mi trabajo es de la persona que tengo a mi lado.

Resumen

El proceso de auscultación dinámica de vías de ferrocarril se basa en la obtención datos sobre las irregularidades geométricas en el carril. Estos datos son de gran importancia para asegurar un correcto funcionamiento de la infraestructura ferroviaria, véase: confort, seguridad, mantenimiento de vehículos, mantenimiento del carril, etc.

En la actualidad la obtención de estos datos supone unos costes elevados debido a la gran cantidad de material necesario para ello. Además, esto supone un control menos riguroso del estado del carril en cada momento, lo que puede producir un desconocimiento por parte del personal encargado del mantenimiento del estado de la vía y puede facilitar que se generen averías.

El objetivo de este trabajo es la consecución de un sistema económico y funcional para la obtención de estos datos, con la particularidad de que pueda ser utilizado en cualquier tren sin más que una pequeña instalación.

Para ello se hace uso de una placa de desarrollo “SmartEverything FOX”, dotada de numerosos sensores y un microprocesador que gestiona los datos recibidos de estos sensores. Además de la SmartEverything, el sistema está dotado de un Arduino con el que esta se comunica y que es el encargado de guardar los datos en una memoria microSD. Estos datos serán después validados mediante la herramienta MATLAB.

El sistema se prueba mediante unos trayectos en bicicleta, debido a la complejidad de realizar pruebas en el entorno del tren. Estas pruebas concluyen con éxito y reportan que el sistema es capaz de reconocer el punto en el que se encuentra y las aceleraciones que se producen en este, lo que da cabida a posibles desarrollos futuros ya sea de esta u otra índole.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

AVE	Alta Velocidad Española
TFG	Trabajo Fin de Grado
MCU	Microcontroller Unit

1 INTRODUCCIÓN Y OBJETIVOS

La infraestructura ferroviaria comprende las instalaciones de señalización, electrificación, mecánicas y de telecomunicaciones en la red de ferrocarril. Toda esta infraestructura requiere de una revisión periódica, la cual se centra en la mejora y mantenimiento del carril ferroviario y toda la estructura que lo comprende para evitar problemas a corto y largo plazo.

Este Trabajo de Fin de Grado se centrará en la problemática del mantenimiento de la parte mecánica de la red de ferrocarril, en concreto del carril de la red de alta velocidad.

1.1 Mantenimiento Ferroviario

En toda obra ferroviaria se encuentran trabajos de mantenimiento y conservación, debido al paso del tiempo la estructura ferroviaria tiende a deteriorarse y para ello sirven este tipo de trabajos. Las actuaciones que se pueden llevar a cabo en los trabajos de mantenimiento y conservación se dividen en:

- Tratamientos preventivos: Son tratamientos que se realizan cada cierto tiempo, con el fin de eliminar daños o deterioros en el carril para garantizar la seguridad de la circulación ferroviaria. Entre estos tratamientos se encuentran algunos como el bateo de desvíos y vías, el reemplazo del carril o el engrase del carril, entre otros. Este tipo de tratamientos son muy corrientes en el mantenimiento ferroviario y evitan que se produzcan errores que precisen de tratamientos correctivos.

Ilustración 1: Máquina Plasser bateando

- Tratamientos correctivos: Son tratamientos cuyo fin es hacer que la vía vuelva al estado normal que tenía antes de producirse un error. Pueden derivar de la falta de tratamientos preventivos, así como de errores inesperados producidos por agentes externos. Este tipo de tratamientos suelen ser más complejos y pueden precisar de un mayor tiempo y costes, lo que a su vez puede provocar interrupciones en el transcurso normal del tren.

Mediante la división entre estos dos tipos de tratamientos podemos llegar a una primera conclusión, los tratamientos preventivos son necesarios para disminuir el número de tratamientos correctivos, aunque estos últimos no pueden ser evitados completamente debido a los agentes externos que puedan actuar sobre la infraestructura ferroviaria.

Actualmente, el mantenimiento ferroviario en España supone una gran problemática debido a los elevados costes asociados al mismo. Esto supone que proyectos de esta índole sean necesarios para la consecución de una reducción de costes, además de provocar una mejora en la calidad del servicio.

1.2 Auscultación Dinámica

El paso del tren por un punto deteriorado o defectuoso del recorrido que realiza provoca una perturbación en la marcha de este y por consiguiente un movimiento del propio tren. Este movimiento no es más que una aceleración horizontal provocada por una hendidura en el terreno o una parte del trazado en mal estado, este tipo de perturbaciones son las que se detectan mediante la auscultación dinámica. La obtención de estos datos proporciona información de cuán maliciosa puede llegar a ser la perturbación en el carril y el tamaño de la avería que supondrá. Tras conocer el tipo de perturbación será necesario conocer el lugar dónde se ha producido para una posterior activación de los protocolos pertinentes a la hora de solucionar o revisar esta incidencia.

1.2.1 Auscultación Dinámica AVE

En la actualidad, concretamente en la línea de alta velocidad, la auscultación dinámica del carril ferroviario (y más elementos que no competen a este trabajo de fin de grado) se realiza mediante el denominado "tren auscultador", también llamado "Tren Séneca", de la serie 330 de ADIF. Este tren, mediante el uso de sistemas de medida y recorriendo el ancho ferroviario, detecta las imperfecciones en el carril y la catenaria. La problemática de este método de auscultación usado en la actualidad es su alto coste, lo que provoca que solo se pueda realizar una auscultación cada uno o dos meses.

Ilustración 2: Serie 330 de ADIF (Tren Séneca) auscultando

El objetivo de este trabajo es diseñar un dispositivo que obtenga esos datos y mediante el cual conseguir la información necesaria del carril en cada momento, en cuanto a lo que auscultación dinámica se refiere. Para ello se tiene en cuenta como principal objetivo la reducción del coste, con el fin de poder aplicarlo en diferentes trenes y poder obtener datos del estado de la vía, como mínimo, a diario.

Gracias a este dispositivo se tendría un mejor conocimiento del estado de cada tramo ferroviario y así, mediante tratamientos preventivos se podrían evitar posibles errores que acarreasen la necesidad de llevar a cabo un tratamiento correctivo.

1.3 Objetivo del Proyecto

Una vez introducida la problemática que se pretende solucionar con este proyecto, se pasa a explicar el objetivo final de este Trabajo Fin de Grado.

El objetivo es el de realizar un dispositivo capaz de detectar las perturbaciones en el carril ferroviario para, posteriormente, analizar estos datos con el fin de encontrar pequeños errores indetectables a simple vista que, solucionándolos, eviten futuras averías de mayor relevancia.

Para ello, el dispositivo debe ser autónomo, de instalación y mantenimiento sencillos, además de económico, ya que esto supondría una mejora sustancial al sistema actual (Tren Auscultador) y que por tanto podría llegar a significar un gran cambio.

Al ser un sistema ya implantado y al que se desea crear una mejora se deben tener en cuenta ciertas consideraciones para que realmente se consiga la mejora esperada:

- El dispositivo debe ser **autónomo**, capaz de trabajar por si mismo y almacenar los datos sin necesidad de agentes externos.
- El dispositivo debe ser capaz de **detectar las perturbaciones** en el trazado y relacionarlas con su **posición** en el recorrido realizado por el tren.
- El dispositivo debe ser capaz de **almacenar datos** de, al menos, un trayecto completo del tren.
- El dispositivo debe ser **económico**, para poder suponer una mejora al sistema que ya hay implantado en la red de ADIF.
- El dispositivo debe ser de **instalación sencilla** en todos los trenes de la red, ya que esto supone una mejoría frente al sistema actual.
- El dispositivo debe ser de **fácil mantenimiento**, ya que la idea es que sea “invisible” para el resto de los elementos que componen la red ferroviaria y solo suponga un apoyo facilitado por el tren para los trabajos de mantenimiento de la red.

1.4 Motivación

Este TFG está motivado por la idea de conseguir un sistema que ayude en la obtención de datos referentes al estado del carril ferroviario y suponga una mejora en la información que tiene en todo momento el puesto de control sobre este. En la actualidad, la baja reiteración del paso del tren auscultador, provoca que haya un desconocimiento del estado del carril durante grandes intervalos de tiempo (de 1 a 3 meses), dependiendo de la cantidad de tiempo que pase entre recorridos de auscultación.

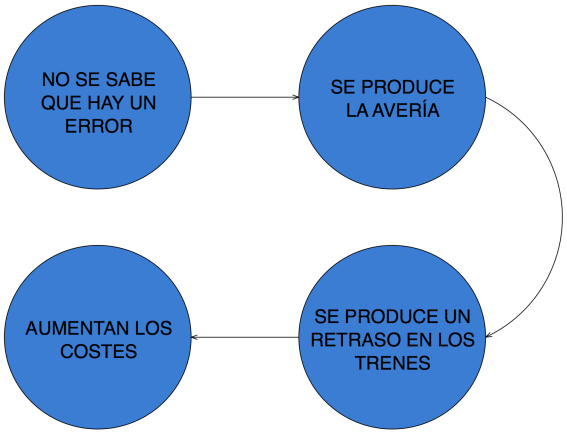

Ilustración 3: Problemática del tren auscultador

Este desconocimiento del estado real del carril ferroviario durante grandes lapsos de tiempo es una de las causas por la que se provocan averías inesperadas, lo que lleva a retrasos en los trenes o cortes temporales del servicio, que a su vez hacen que aumenten los costes, tanto de mantenimiento, por las horas extra requeridas a los trabajadores encargados de ello y por el material requerido para el mismo, como los costes efectuados en modo de compensación económica a los usuarios afectados por el retraso en su viaje.

Gracias a una mayor información sobre el estado de la vía será posible conseguir un mayor control sobre el mantenimiento de esta, lo que puede suponer una mejora de: el confort de los usuarios, la duración del carril debido a su menor desgaste y una disminución de costes de mantenimiento a largo plazo.

2 DESARROLLOS EXISTENTES

Este apartado se centrará en el estudio de los diferentes usos que se han dado y se dan en la actualidad a los sistemas con los que trabaja este TFG. De este modo, se divide en cuatro apartados principales que suponen las cuatro partes esenciales del proyecto: medidas con acelerómetro, uso de la localización GPS, entorno y sistema Arduino y sus variantes y por último auscultación dinámica.

En los apartados venideros se tratarán estas cuatro partes haciendo referencia a proyectos o desarrollos ya existentes y en los que se apoyan ciertos aspectos de este TFG.

2.1 Medidas con Acelerómetro

Un acelerómetro, es un dispositivo capaz de medir aceleraciones, es decir, la variación de la velocidad por unidad de tiempo. Hay distintos tipos de acelerómetros: mecánicos, piezoeléctricos, de condensador, etc. Esto supone que se puedan utilizar en diferentes escenarios y con distintos fines según las características de cada uno.

El acelerómetro es uno de los dispositivos más utilizados en la actualidad, desde su primera comercialización en **1923** [1], ha pasado a formar parte de elementos como:

- **Aviación:** El acelerómetro es de suma importancia en aviónica, ya que es muy importante en todo momento conocer la referencia inercial del avión. Esto misma pasa en un campo relevante de la actualidad como es el caso de los drones, campo en continuo crecimiento y que precisa de acelerómetros y giróscopos para mejorar la estabilización y el posicionamiento del dron en todo momento. En el caso de la aviónica actual esto se lleva a cabo mediante un equipo de giróscopos y acelerómetros [2] anclados al avión como el siguiente:

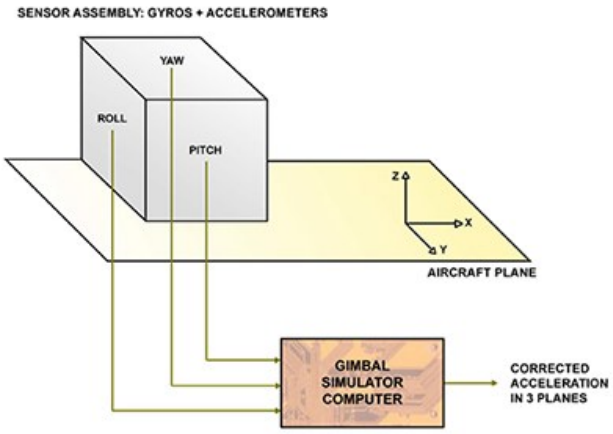

Ilustración 4: Sistema Inercial de Aviación

- **Medicina:** El campo médico es otro de los que se nutre del uso de acelerómetros en múltiples y diferentes aplicaciones. Uno de los campos en crecimiento actualmente en el marco de la medicina es el de la monitorización de la actividad física. Aunque en la actualidad se está produciendo un aumento del mercado de las pulseras de actividad física, también constituidas por acelerómetros, este tipo de estudios de actividad de manera médica [3] se lleva a cabo mediante un cinturón a la altura de la cintura que contiene el acelerómetro y que es capaz de, mediante una electrónica complementaria, dar el cálculo de la actividad física diaria de una persona.

- **Smartphone:** Otro campo en el que los acelerómetros son una referencia clave es el campo de la telefonía. El acelerómetro es el encargado de detectar los cambios de orientación en los terminales, esto permite gestionar la posición vertical u horizontal de la pantalla, realizar seguimiento de la actividad física o calcular distancias, entre numerosas aplicaciones más.

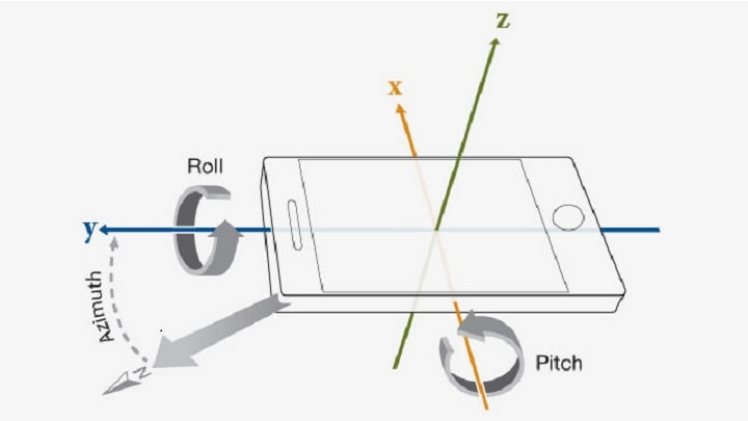

Ilustración 5: Acelerómetro en Smartphone

Según datos de 2017, el número de usuarios de móviles se elevaba por encima de los 5000 millones, teniendo esto en cuenta se puede llegar a comprender la importancia de los acelerómetros en nuestro día a día.

- Además de estas aplicaciones, podemos encontrar publicaciones como "**Modelos para la generación y transmisión de vibraciones al paso de un tren**" [4] de cuatro ingenieros e ingenieras de la Universidad Politécnica de Cataluña, que sostienen, en cierto modo, el objetivo de este TFG.

  En esta publicación se encuentran definidas la modelización de la vía, de la fuerza de contacto, respuesta dinámica de la vía y terreno. Para la adquisición de datos y obtención de las medidas se propone usar un acelerómetro bajo el carril ferroviario, dando especial importancia a las aceleraciones verticales, al ser estas las más significativas.

Ilustración 6: Disposición de acelerómetro bajo carril ferroviario

- Por último, como se mencionó en el primer apartado, el tren auscultador al que se pretende sustituir con este dispositivo es lo más parecido al objetivo que tiene este proyecto, ya que realiza funciones similares a las que lleva a cabo este dispositivo.

  El tren auscultador, concretamente el tren auscultador Séneca, fue desarrollado por la Dirección de Innovación Tecnológica de Adif, junto con Talgo. El Séneca realiza la auscultación dinámica de la vía y catenaria de la red de alta velocidad y para ello cuenta múltiples dispositivos cuyo eje principal son los acelerómetros, que proveen al tren auscultador de la capacidad necesaria para detectar las irregularidades del carril.

Ilustración 7: Tren Séneca

2.2 Uso del GPS

El Sistema de Posicionamiento Global (GPS; Global Positioning System), es un sistema mundialmente conocido que permite determinar la posición de un objeto en todo el planeta Tierra con una precisión muy alta.

El sistema GPS funciona mediante una red de satélites en órbita con trayectorias sincronizadas para cubrir toda la superficie terrestre. Cuando se desea obtener la posición de un objeto, el receptor localiza como mínimo cuatro satélites, de estos recibe su identificación y la hora, además de información sobre la constelación. Basándose en estos datos el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, así, mediante el modo de trilateración inversa calcula la distancia al satélite. Conocidas estas distancias y la posición de cada uno de los satélites se obtienen las coordenadas del receptor.

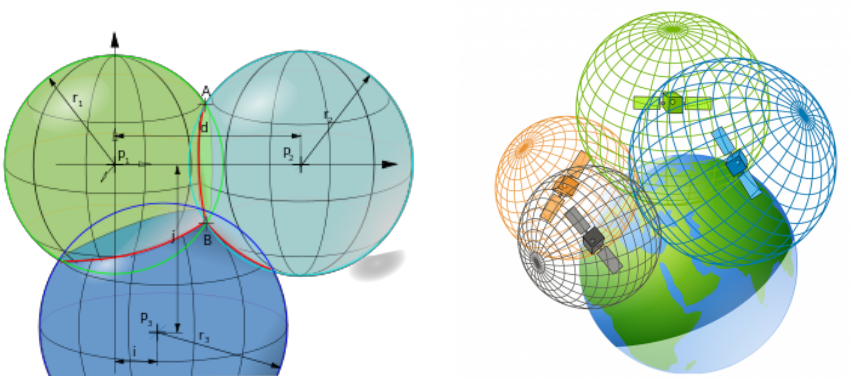

Ilustración 8: Proceso geométrico para determinar la posición mediante trilateración.

El sistema GPS ha producido desde su llegada un cambio radical en el día a día de la sociedad, gracias a la facilidad de conocer en cualquier momento la posición de un punto. Esto ha sido posible gracias a su integración en sistemas como los Navegadores GPS o los Smartphones, pero el GPS no solo se queda ahí, sino que ha proporcionado una gran ayuda en proyectos muy diferentes entre sí como:

- **Movimiento de placas tectónicas:** Desde 1994 [5], el Consejo Nacional de Investigaciones Científicas y Tecnologías de Venezuela (CONICIT) lleva a cabo una serie de mediciones GPS en varios sitios localizados en las Antillas Menores, Trinidad y Venezuela, que comparan entre sí para comprobar el movimiento que se produce en las placas tectónicas del Caribe y Suramérica. Gracias a este estudio se puede llevar a cabo un análisis de la velocidad a la que se mueven estas placas y la tasa anual de desplazamiento de dichas placas, lo que permite cuantificar el riesgo sísmico y prevenir de ello a los países afectados.

- **Seguimiento de la fauna en peligro de extinción:** El Programa de Seguimiento de Procesos y Recursos Naturales de Doñana (PSD) nace como respuesta a la necesidad de los gestores de este espacio protegido de tomar decisiones en cuanto a la conservación y mejora del mismo y de su hábitat. El Espacio Natural de Doñana es una reserva natural situada al suroeste de península, ocupando en su mayor parte la provincia de Huelva y en menor parte las provincias de Sevilla y Cádiz. En este espacio natural se concentra una gran cantidad de fauna y flora, mucha de ella protegida o en peligro de extinción como es el caso del Lince Ibérico. Debido a la búsqueda de la reproducción y aumento de este tipo de especies es necesario tener un control sobre su localización y su forma de interactuar con el entorno que les rodea, para ello se utilizan dispositivos GPS que llevan consigo los animales a los que se desea localizar. Estos dispositivos GPS son implementados con secuencias específicas para cada tipo de muestreo, mediante el programa *Cybertracker*.

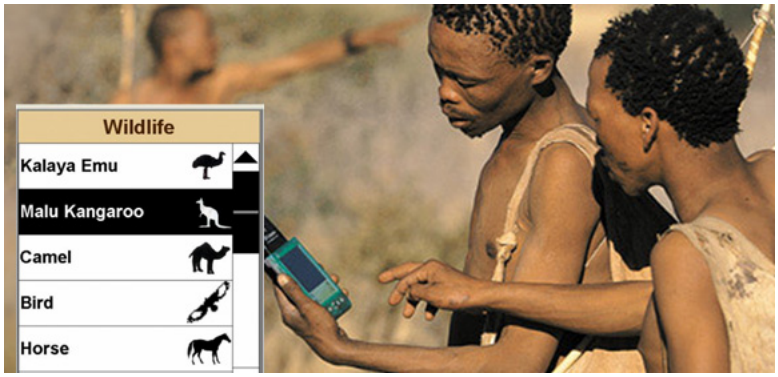

Ilustración 9: Herramienta de seguimiento *Cybertracker*

El programa Cybertracker comenzó a implementarse en tribus indígenas africanas para el seguimiento del rinoceronte negro, siguiendo sus movimientos y comportamientos al minuto. Hoy en día, la empresa distribuye una herramienta software utilizada tanto para animales, personas, climatología, etc. El objetivo es conseguir desarrollar una herramienta de monitoreo medioambiental a nivel mundial.

- Un proyecto más cercano al objetivo que se abarca en este TFG es "**Prototipo para la Monitorización de la Infraestructura Ferroviaria utilizando sensores LIDAR, Imagen de Vídeo y GPS**"[6], el cual lleva a cabo la monitorización en tiempo real de la línea de Alta Velocidad de ADIF mediante datos recibidos de sensores láser, cámara de vídeo y un receptor GPS.

  El sensor LIDAR va recogiendo durante el trayecto la distancia a los obstáculos en un ángulo de hasta 360º, gracias a tener un módulo GPS acoplado que registra las coordenadas de los puntos recorridos se puede realizar una reconstrucción 3D del entorno de la vía férrea.

  El punto negativo de este sistema es la necesidad de instalar un sistema de gran tamaño y coste en el tren, como el que se ve en la siguiente imagen:

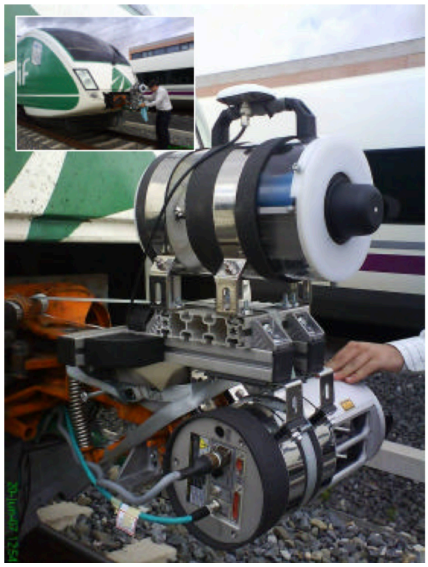

Ilustración 10: Sensores instalados en el tren

2.3 Arduino

Arduino es una plataforma de electrónica de código abierto, con software y hardware sencillo y respaldada una extensa comunidad. Arduino diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos capaces de interactuar con todo tipo de sensores. Generalmente el hardware consiste en un microcontrolador **Atmel AVR**, sobre una placa de circuito impreso a la que se conectan placas de expansión a través de la disposición de los puertos de entrada y salida presentes en la placa. Al ser software y hardware diseñado en conjunto para realizar todo tipo de proyectos, se asegura compatibilidad y sencillez en el desarrollo de proyectos en este entorno.

- **Software Arduino:** El software ofrecido por Arduino consta de un entorno de desarrollo integrado (IDE) así como de numerosas librerías, creadas tanto por la propia empresa Arduino como por la comunidad de personas que desarrollan en este tipo de placas.

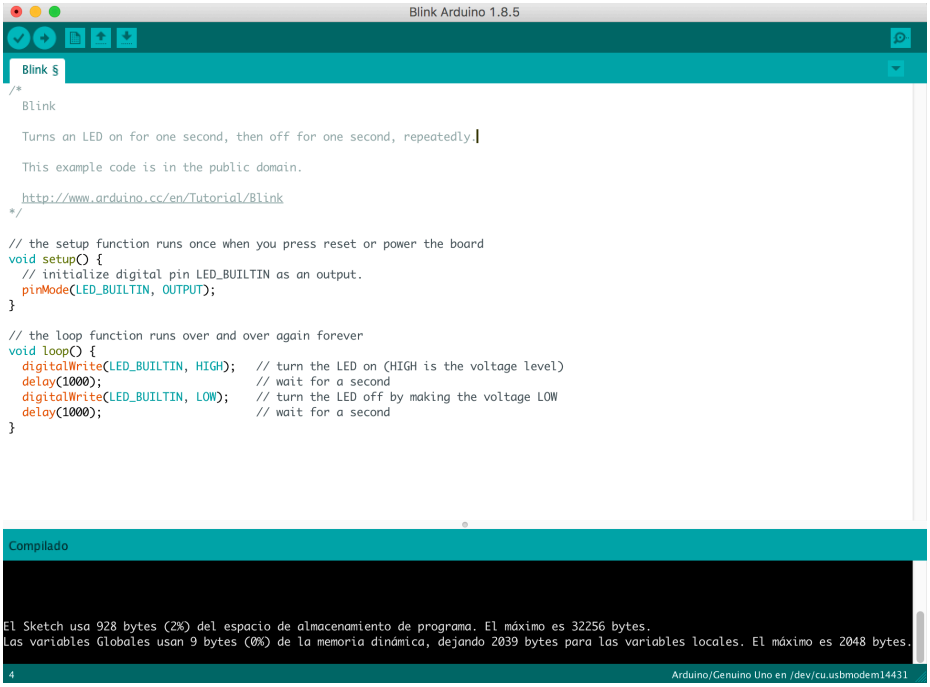

Ilustración 11: Entorno Arduino IDE

- **Hardware Arduino:** El hardware Arduino se compone de infinidad de placas, ya que al ser hardware *Open-source* cualquier persona puede realizar una. Aún así, Arduino distribuye sus propias placas y con ciertas diferencias de tamaño y microprocesadores entre ellas con el fin de que cada usuario obtenga la más adecuada para su proyecto.

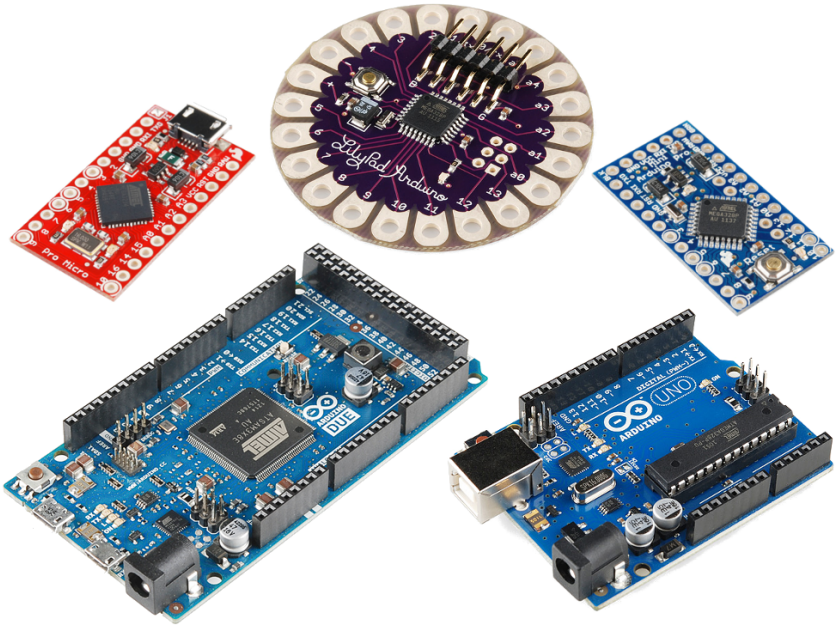

Ilustración 12: Diferentes placas Arduino

Arduino permite desarrollar tanto elementos autónomos como conectarse con otros dispositivos y elementos para conseguir objetivos de mayor envergadura. Son placas de desarrollo con una gran adaptabilidad y que suponen un apoyo importante tanto en proyectos universitarios como en proyectos de la industria.

Además de la propia industria Arduino, se llevan a cabo desarrollos paralelos basados en la misma, como se ha comentado anteriormente que suponen un cambio y en ocasiones un paso más en la trayectoria de Arduino. Un ejemplo de este tipo de placas desarrolladas de forma independiente es ***Ruggeduino***, un proyecto desarrollado por *Rugged Circuits* [4] y que consiste en una placa compatible con Arduino UNO tanto en hardware como en software.

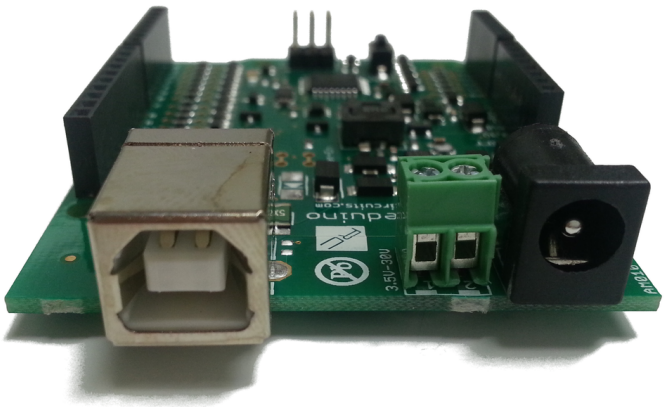

Ilustración 13: Placa de desarrollo *Ruggeduino*

La característica principal y que hace diferencial esta placa es que reduce enormemente su sensibilidad frente al ruido eléctrico. Uno de los problemas que más afectan a las placas Arduino es la sensibilidad al ruido en ambientes con alto ruido eléctrico, esto llevó a *Rugged Circuits* a desarrollar esta placa y conseguir así solucionar uno de los mayores inconvenientes de Arduino.

En la Universidad de Uppsala en Suecia se llevó a cabo un estudio con Arduino basado en medir vibraciones mecánicas [7], para demostrar cómo estas pueden afectar a las medidas y por tanto al resultado de un experimento. Para este estudio se utilizó un Arduino UNO y dos acelerómetros MEMS para captar las medidas. Primero se realizó un estudio controlado mediante vibraciones provocadas por un altavoz, que al generar un tono conocido permitía a los investigadores saber si la medida era correcta o no. Una vez pasada esta prueba se probó en un entorno real de laboratorio, colocando los acelerómetros sobre dos partes distintas de una bomba de vacío y se obtuvieron resultados como el siguiente:

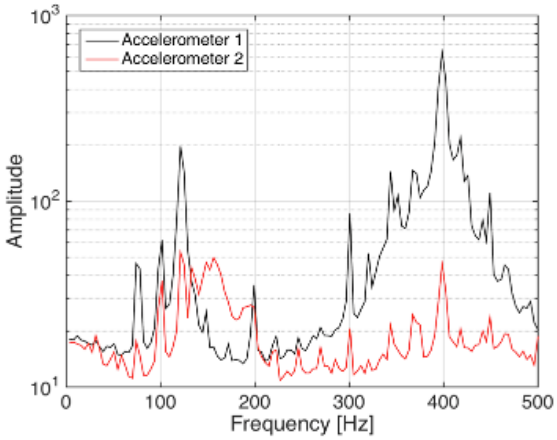

Ilustración 14: Datos de vibración sobre bomba de vacío

Gracias a la propia placa Arduino y al software que se explicó anteriormente y apoyándose de la herramienta MATLAB, se pudo llevar a cabo este estudio que determinó que con hardware y software de bajo precio se podía realizar un sistema de adquisición de datos con grandes resultados.

3 DESARROLLO DEL PROYECTO

En este apartado se detallará extensamente la problemática a solucionar y el desarrollo que se ha llevado a cabo para conseguirlo, además se detallarán los componentes que se han utilizado para este desarrollo, la programación llevada a cabo y un breve resumen de pruebas parciales que se han realizado.

Como se ha detallado en anteriores apartados, la problemática principal en la que se centra este Trabajo de Fin de Grado es la de conseguir detectar las perturbaciones verticales en el trazado del carril ferroviario. Aunque esto ya se lleva a cabo mediante el tren auscultador, el objetivo principal de este proyecto es conseguir un dispositivo pequeño, de bajo coste y que sea de fácil implementación en cualquier tren, de modo que se elimine la problemática de tener un tren encargado exclusivamente de esta tarea y aprovechando así el paso de los trenes convencionales por la vía diariamente.

3.1 Planteamiento del problema

El paso del tiempo y los agentes externos, provocan un movimiento del balasto que sujeta el carril y un desgaste de las vigas o del propio carril, en definitiva, generan un deterioro en el entramado sobre el que circula el tren.

Ilustración 15: Carril ferroviario y balasto sobre el que se apoya

Debido a ello, es necesario realizar el control exhaustivo del trazado de la vía sobre la que circulan los trenes y este es el objetivo que se busca conseguir.

En definitiva, el problema que se presenta en este TFG es el de medir las aceleraciones horizontales de un tren en marcha a, como máximo, 300 km/h y localizar la posición de estas aceleraciones dentro del recorrido tren. Además, se desea que estas aceleraciones se almacenen para poder, tras el viaje, realizar un estudio detallado de las mismas para una posterior actuación sobre ellas dependiendo del nivel de gravedad de estas.

Para conseguir este objetivo se divide el problema en **tres partes** principales: medida de la aceleración, medida de la posición y almacenamiento y procesamiento de los datos.

3.1.1 Medida de la aceleración

Las aceleraciones a medir para la consecución del objetivo son las del movimiento horizontal del tren provocadas por las imperfecciones en el carril. Para conocer numéricamente las aceleraciones de las que hablamos con el fin de elegir correctamente la tecnología a utilizar se llevan a cabo una serie de cálculos.

Teniendo en cuenta la trayectoria del tren en sentido horizontal, el esquema de fuerzas quedaría tal que así:

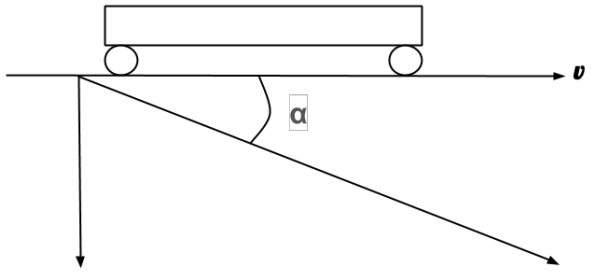

Figura 1: Esquema de fuerzas movimiento del tren

La ecuación principal que define el movimiento del tren durante un bache es:

$$a_z = \frac{\Delta v_z}{\Delta t} = \frac{v \cdot \mathrm{tg}\alpha}{\frac{l/2}{v}} = \frac{2v^2 \cdot \mathrm{tg}\alpha}{l}$$

Ecuación 1

Donde:

l: Longitud del bache

v: Velocidad del tren

α: Ángulo que forma la trayectoria horizontal y la que seguiría el tren durante el bache

Extrayendo información de datos oficiales de las gráficas de auscultación dinámica del tren Séneca, se conoce que los desperfectos, o baches, que se pueden dar durante el trayecto del tren pueden ir entre 0.5cm y 2cm de profundidad y 0.6m y 5m de largo. Por tanto, tomando α como el arcoseno de la profundidad del bache y los valores más limitantes de los anteriormente citados se obtiene:

$$a_z = \frac{2v^2 \cdot tg\alpha}{l} = \frac{2 \cdot 83.33^2 \cdot \mathrm{tg}(\sin^{-1}(0.02))}{0.6} = 463.019\ \frac{m}{s^2} = 47.246g$$

Por tanto, la aceleración máxima que ha de medir el dispositivo serán $47.246g$, aunque esto es suponiendo una imperfección en el terreno que se dará en muy pocas ocasiones debido a la pequeña longitud del bache que se ha considerado y que con un aumento leve ya supondría una aceleración a medir mucho menor.

3.1.2 Medida de la posición

La medida de la posición es un proceso más sencillo en cálculo respecto a la medida de la aceleración, ya que lo que recibimos del receptor GPS serán directamente coordenadas en forma de latitud y longitud. El GPS deberá registrar correctamente y en todo momento los cambios de posición que se efectúen en el trayecto del tren.

Conociendo que la velocidad máxima permitida para el tren son 300km/h, y llevando a cabo el siguiente cálculo:

$$300\ {}^{km}/_{h} \cdot \frac{1000\ m}{1\ km} \cdot \frac{1\ h}{3600\ s} \cdot \frac{1\ s}{1000\ ms} = 0.0833\ {}^{m}/_{ms}$$

Ecuación 2

Se puede ver cómo con una tasa de refresco de la posición de 1ms sería suficiente para nuestra aplicación, ya que se tendría más información incluso, que la necesaria.

3.1.3 Almacenamiento y procesamiento de los datos

Para el almacenamiento de los datos será necesario el apoyo de una fuente externa de memoria o tener una gran cantidad de memoria en la propia placa en la que se desarrolle el proyecto, ya que durante una travesía se han de almacenar cientos de miles de datos tanto de posición (latitud y longitud) como de aceleración. Para ello lo más conveniente sería el uso de una memoria externa tipo SD o similar.

La parte de procesamiento de los datos sería una de las más interesantes del trabajo, ya que la idea final sería obtener algo parecido a la siguiente gráfica:

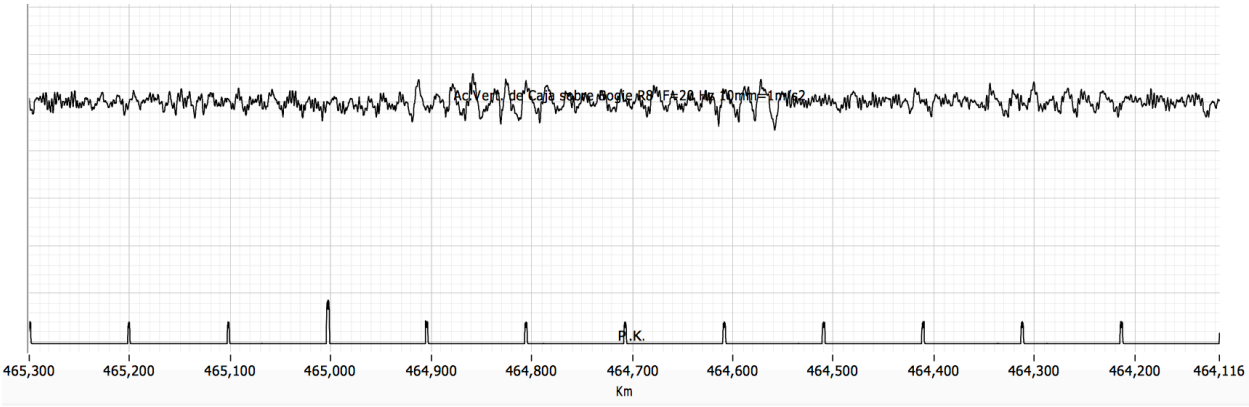

Ilustración 16: Gráfica tren Séneca

Esta gráfica es un extracto de la información que proporciona el tren Séneca en la actualidad y que ofrece información de la aceleración vertical en todo el recorrido, dando como referencia en este caso los P.K (puntos kilométricos) a diferencia de latitud y longitud como se darán en este TFG. Una vez obtenidas estas gráficas se han de tener en cuenta los rangos entre los cuáles puede estar una aceleración para ser considerada como un error en la trazada:

1. Corrección Inmediata: Ac > 70 m/s^2
2. Corrección Programada: Ac > 50 m/s^2
3. Seguimiento: Ac > 30 m/s^2

Estos datos son los datos reales usados en la auscultación actual.

En el primer caso, Corrección Inmediata, será necesaria la atención de la incidencia de inmediato por parte de un técnico, ya que este tipo de imperfecciones pueden producir accidentes tales como descarrilamientos en un corto periodo de tiempo. Este tipo de errores son los más importantes y por tanto en los que más hincapié se hará para encontrarlos.

El segundo caso, Corrección Programada, serán los casos que están en un nivel intermedio, habría que llevarlos a análisis por parte del personal encargado para conocer si pueden llegar a ser peligrosos a corto plazo o necesitan simplemente un seguimiento que asegure el buen funcionamiento.

Y, por último, los casos de seguimiento. Son los más comunes ya que se refieren a unas aceleraciones relativamente bajas y que, a veces, pueden incluso ser inapreciables a la hora de revisarlas, de ahí que solo requieran un seguimiento por parte del personal encargado con vistas a conocer si pasan a un estado de mayor complejidad.

Para obtener este tipo de gráficas y rangos de valores será necesario combinar aceleración y posición, de modo que cada punto de latitud y longitud quede reflejado con un valor de aceleración. Además, se habrá de hacer una media de los valores de aceleración, ya que, al producirse grandes vibraciones en la trayectoria del tren, sin una media se tendería a errar a la hora de analizar los datos y estos podrían dar a equívocos.

3.2 Elección de componentes

La elección de componentes supone uno de los elementos más importantes a la hora de realizar un proyecto en este ámbito. En el caso de este Trabajo Fin de Grado es el elemento principal, ya que una buena elección de componentes y elementos que compondrán el dispositivo final supone la consecución final de todos los objetivos planteados en el anterior apartado. Para cumplir cada uno estos objetivos se debe hacer uso de ciertos componentes que serán detallados a continuación.

3.2.1 Componentes para la medida de la aceleración y la posición

Una vez vistas las cualidades que ha de tener nuestro dispositivo en materia de medida de aceleraciones y posición, se llega a la conclusión de que los elementos principales con los que ha de contar son:

- **Receptor GPS:** Para medida de la posición.
- **Acelerómetro:** Para medida de las aceleraciones en, al menos el eje X.
- **Microprocesador:** para el cálculo y la obtención de los datos de salida.

Por otra parte, sería interesante contar con otros elementos como:

- **Conexión inalámbrica:** Permitiría obtener los datos de forma inalámbrica y en directo, lo que podría ser beneficioso a la hora de detectar averías repentinas.
- **Ser reprogramable:** Un dispositivo que permitiese modificar su programación en un futuro permitiría añadir mejoras o cambios que supusiesen un aumento en las prestaciones de este.

Por tanto, tras un estudio del tipo de dispositivos que se podían usar para este fin, y gracias a la disponibilidad en el departamento de este tipo de placas, se decide usar la placa de desarrollo *"SmartEverything FOX"*, programable en entorno Arduino.

La *SmartEverything FOX* es una placa de desarrollo que cuenta con el microcontrolador ***Atmel D21*** y que, gracias al soporte del IDE de Arduino, facilita la programación y por tanto el desarrollo de todo tipo de proyectos.

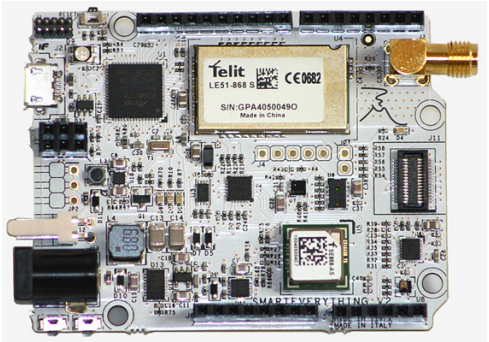

Ilustración 17: SmartEverything FOX

Además de contener gran cantidad de periféricos, como se detallará a continuación, esta placa de desarrollo posee, al igual que Arduino, de una serie de pines de entrada y salida. Estos pines facilitan la interconexión de la placa con otros sensores, placas de desarrollo y todo tipo de elementos para hacerla, si cabe, más completa.

1. ATMEL SAMD21:

El SAMD21 es un microcontrolador que usa el procesador de 32-bit ARM Cortex-M0 + MCU y que es el centro de todas las operaciones de la placa.

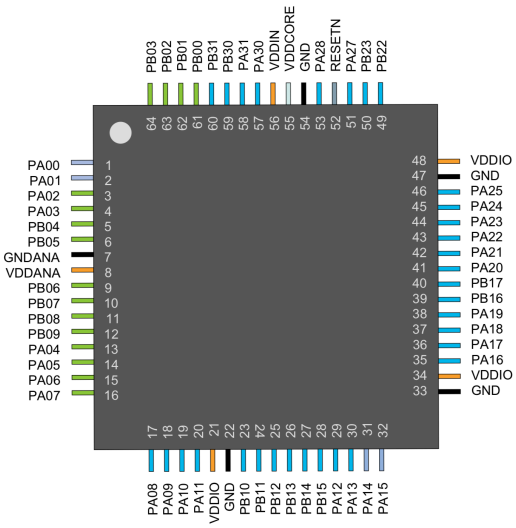

Ilustración 18: Pinout ATMEL SAMD21J

Del microcontrolador que acompañará a la placa cabe destacar:

Tabla 1: Características principales *ATMEL SAMD21J*

Flash programable 256KB
Memoria principal SRAM 32KB
Memoria de bajo consumo SRAM 12KB
16 canales de acceso directo a memoria (DMAC)
Hasta 5 Timers/Contadores configurables de 16-bits
3 Timers/Contadores para control de 16-bits
Contador en tiempo real de 32-bits con función de reloj/calendario
Watchdog Timer
Interfaz USB 2.0
1 conversor analógico-digital (ADC) de 12-bits y hasta 20 canales
1 conversor digital-analógico (DAC) de 12-bits
2 comparadores analógicos (AC)
Tres amplificadores operacionales (OPAMP)

Además, el microcontrolador se conecta con numerosos sensores y periféricos que cuentan con sus propias librerías. A diferencia de Arduino, no precisa de conexiones a módulos externos de sensorización por parte del usuario. Entre sus distintos periféricos se encuentran:

- Atmel Crypto Authentication chipset.
- Antena Dynaflex 868Mhz.
- Módulo Sigfox.
- Módulo GPS.
- Sensor de proximidad.
- Sensor de humedad y temperatura.
- Sensor axial.
- Sensor de presión.
- Chip NFC.
- Interfaz Bluetooth.

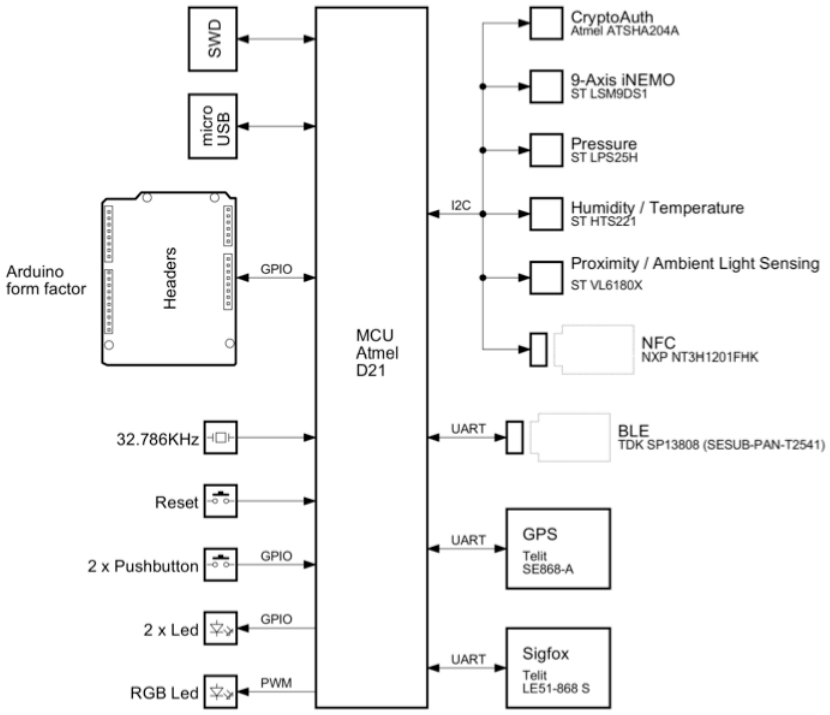

Ilustración 19: Diagrama de bloques SmartEverything

De estos periféricos, no se usarán todos, aunque sí que se pueden tener en cuenta algunos que no se han usado para futuras mejoras. Los necesarios para el primer desarrollo del TFG serán el módulo de acelerómetro y el módulo de GPS.

2. Módulo acelerómetro LSM9DS1:

El módulo acelerómetro, por su parte, será el encargado de medir las aceleraciones que se produzcan en todo momento en el trayecto del tren, siendo de principal interés las que superen el umbral calculado en el punto **3.1.1**. El **LSM9DS1**, acelerómetro que monta la SmartEverything, es un sensor de aceleración 3D lineal, angular y magnético. Se comunica con la MCU a través de interfaz I2C.

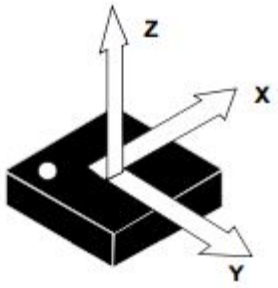

Ilustración 20: Acelerómetro LSM9DS1

Entre los elementos más importantes a destacar del LSM9DS1 están:

Tabla 2: Características principales *LSM9DS1*

3 canales	Aceleración lineal, velocidad angular y campo magnético			
Sensibilidad de aceleración lineal	±2g = 0.061 mg/LSB	±4g = 0.122 mg/LSB	±8g = 0.244 mg/LSB	±16g=0.732 mg/LSB
Sensibilidad magnética	±4g = 0.14 mgauss/LSB	±8g = 0.29 mgauss/LSB	±12g = 0.43 mgauss/LSB	±16g = 0.58 mgauss/LSB
Sensibilidad de velocidad angular	±245 dps = 8.75 mdps/LSB	±500 dps = 17.50 mdps/LSB	±2000 dps = 70 mdps/LSB	
Salida de datos	16-bit			
Interfaces serie	SPI / I^2C			
Alimentación analógica	1.9 V a 3.6 V			

3. Módulo GPS SE868-A:

El módulo GPS Telit Jupiter SE868-A, será el que dicte las posiciones GPS del tren en todo momento y que ayude a conocer el lugar geográfico de las zonas conflictivas del trayecto. Con soporte GPS, QZSS, GLONASS y compatible con el satélite Galileo, este periférico tiene una antena SMT embebida y es capaz de rastrear constelaciones GPS y GLONASS (y eventualmente Galileo) simultáneamente y proveer de posición a través de una interfaz serie estándar (UART).

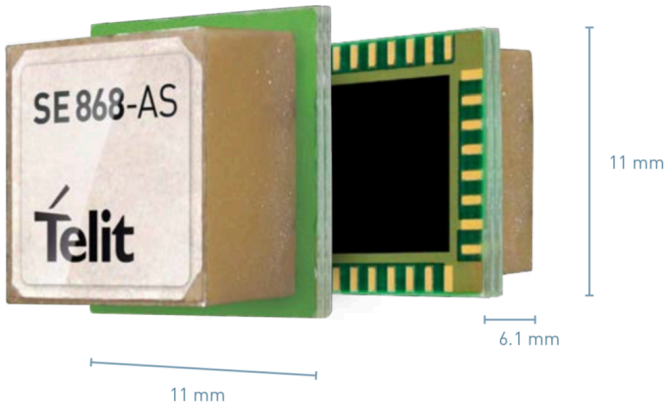

Ilustración 21: Módulo GPS SE868-A

Las características a destacar del SE868-A son:

Tabla 3: Características principales *SE868-A*

Bandas de Frecuencia	GPS L1, QZSS L1		
Estándares	NMEA		
Interfaz	UART		
Precisión posicional	Error < 3m		
Alimentación	2.8 V a 4.3 V		
Sensibilidad	Adquisición: -148dBm	Navegación: -163dBm	Tracking: -165dBm

Estos serán los elementos principales de la SmartEverything, pero no serán todos los utilizados en el proyecto, ya que, como se ha explicado anteriormente, se precisa de una fuente de almacenamiento externa para los datos, debido a la escasa memoria de la placa de desarrollo. Para ello se utilizarán otros elementos que se conectarán a la SmartEverything mediante sus pines de entrada y salida y que supondrán una mejora y el complemento adecuado para la consecución del objetivo de este TFG.

3.2.2 Dispositivos de almacenamiento y procesamiento de los datos

Para poder guardar los datos recibidos por acelerómetro y GPS será necesario un dispositivo de almacenamiento externo, en este caso se opta por una tarjeta microSD. Para esto, se precisa de un modulo externo de lectura y escritura que sea compatible con nuestra placa.

Aquí se encontró un pequeño conflicto en el desarrollo del TFG, ya que se disponía en el departamento de un módulo para lectura y escritura microSD como el siguiente:

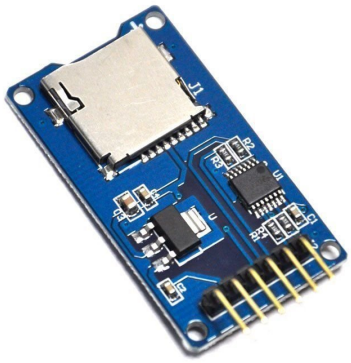

Ilustración 22: Módulo lectura/escritura microSD disponible en el departamento

La disponibilidad de este hizo plantear desde un primer momento. A priori, esto parecía posible ya que al ser un módulo compatible con Arduino no era de extrañar que también lo fuese con la SmartEverything, pero, al ir acompañada esta del microprocesador *ATMEL SAMD21*, a diferencia del Arduino UNO (con el que es compatible el módulo) que monta el *ATmega328P*, no fue posible hacer escrituras en la tarjeta microSD mediante el módulo.

Este problema planteó dos opciones, llevar a cabo la búsqueda de un nuevo módulo de lectura/escritura microSD compatible con el microcontrolador de la SmartEverything o una segunda vía que se basaba en el uso de un Arduino UNO conectado directamente a la SmartEverything y el cual fuera el encargado de leer/escribir en la tarjeta microSD mediante el módulo externo.

A continuación, los pros y contras de cada variante:

Tabla 4: Pros y Contras adición Arduino UNO

	PROS	CONTRAS
Nuevo módulo compatible	Menor coste, programación unificada en una única placa	Poca velocidad de recolección de datos debido a la necesidad de parar para almacenar los datos
Uso de Arduino UNO	Separación entre el procesamiento de los datos y el almacenamiento	Mayor coste, aumento del número de errores en la codificación al tener un código en cada placa

Debido a la disponibilidad del Arduino UNO, lo que hacía eliminar el sobrecoste del uso del mismo y teniendo en gran consideración las facilidades y mejora en velocidad que supondría separar procesamiento de datos y almacenamiento en dos placas diferentes se decide optar por la vía del uso de un Arduino UNO. Para establecer la conexión entre ambas placas se decide usar el protocolo I^2C por su facilidad de uso (solo precisa de dos cables) y porque la velocidad de transmisión de los datos que ofrece es suficiente para esta conexión.

4. Módulo de escritura/lectura microSD WODE:

Este módulo es el encargado de la lectura y escritura en la tarjeta microSD. La conexión entre el Arduino y el módulo se realiza mediante el protocolo SPI (Serial Peripheral Interface).

El SPI es un protocolo de comunicación síncrono usado para la transferencia de información entre circuitos integrados en equipos electrónicos, en este caso en Arduino y el módulo microSD.

Para la sincronización y transmisión de los datos el módulo y el Arduino usan los 4 pines:

- **CS** (*chip select*): Mediante este pin el maestro avisa al esclavo de su activación.
- **SCK** (*señal de reloj del maestro al esclavo*): Es el pulso mediante el que se realiza la sincronización. Con cada pulso de este reloj, se lee o se escribe un bit.
- **MOSI** (*Master Out Slave In*): Es la salida de datos del maestro y entrada de datos del esclavo.
- **MISO** (*Master In Slave Out*): Es la salida de datos del esclavo y entrada de datos del maestro.

Además, son necesarios los 2 pines restantes:

- **VCC** (*alimentación*): A 3.3V o 5V.
- **GND** (*tierra*): Tierra del sistema.

Cuando se desea comenzar una transmisión el maestro pone a 0 el CS, el esclavo se activa y comienza el envío de los datos. Se envía la cadena de bits de forma síncrona mediante los pulsos de reloj, con cada pulso de reloj el maestro envía un bit al esclavo.

La conexión Arduino-Módulo quedaría tal que así:

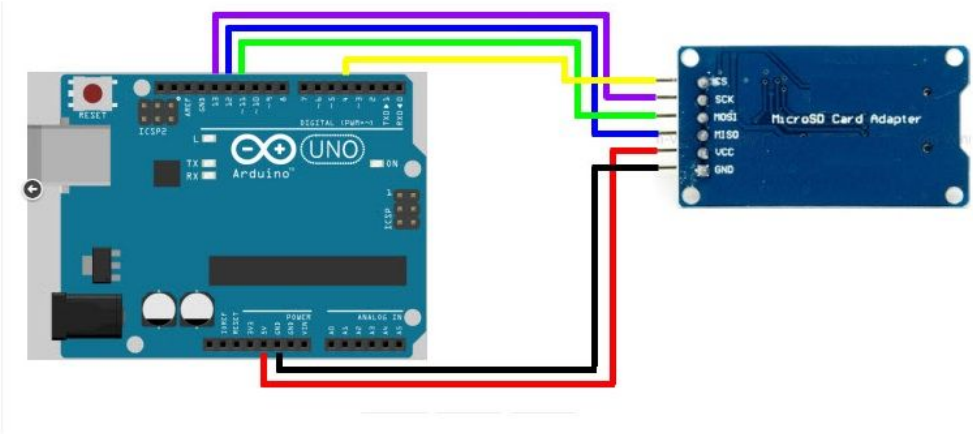

Ilustración 23: Conexión Arduino UNO y módulo microSD

5. Arduino UNO:

El Arduino UNO, junto al MEGA, supone el mayor número de ventas de placas de desarrollo Arduino. Es la placa de desarrollo con la que la mayoría de las personas interesadas en el desarrollo se introducen en este ámbito. El Arduino UNO se usa para construir diferentes circuitos electrónicos usando una placa programable con el microcontrolador *ATmega328P* y un código programado en una versión simplificada de C++, mediante la conexión USB a un ordenador.

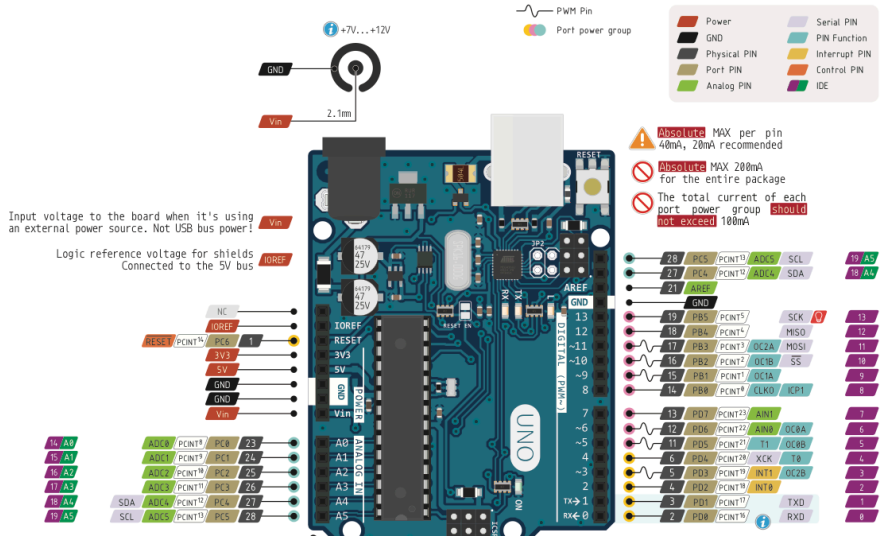

Ilustración 24: Pinout del Arduino UNO Rev3

La placa tiene 14 pines digitales de entrada/salida (de los cuales 6 pueden ser usados como salida PWM), 6 entradas analógicas, un oscilador de 16Mhz, conexión USB, un jack de alimentación, un cabezal ICSP y un botón reset.

Como se ha mencionado, el elemento central del Arduino UNO es su microcontrolador *ATmega328P* el cual presenta las siguientes características principales:

Tabla 5: Características principales *ATmega328P*

28/32 Pines
Memoria Flash 32KB
Memoria SRAM 2KB
Memoria EEPROM 1KB
23 líneas de entrada/salida de propósito general
2 SPI
1 I^2C
1 USART
Interfaz USB 2.0
8 convertidores analógico-digital (ADC) de 10-bits
2 timers/contadores de 8-bits
1 timer/contador de 16-bits

Siendo estos todos los componentes, a continuación, un esquemático de como quedaría el sistema completo interconectado:

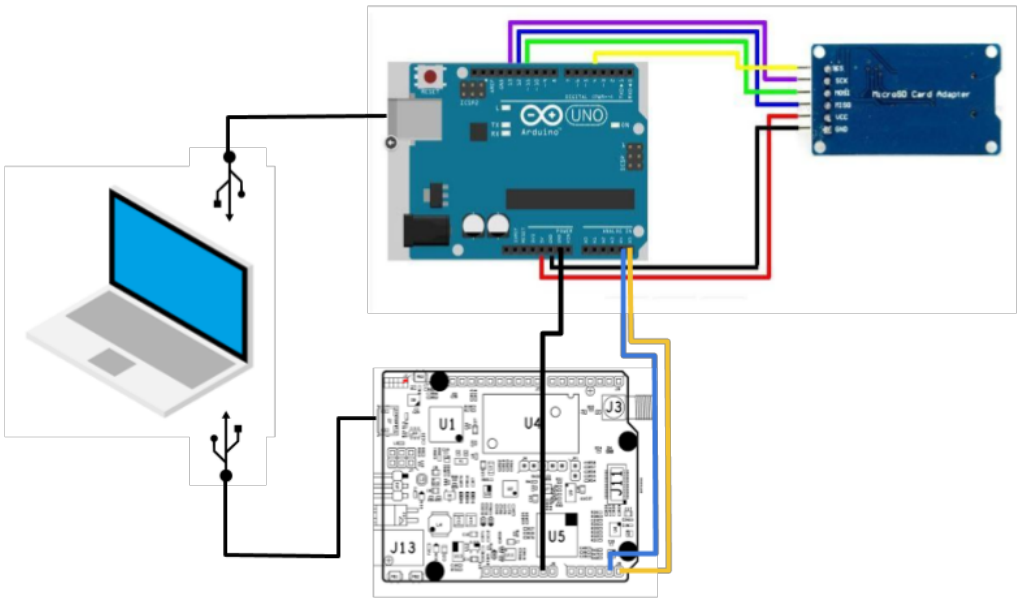

Ilustración 25: Esquemático sistema completo

3.3 Programación

Una vez presentados los componentes elegidos en este TFG se pasa a hablar de la programación de los componentes, que supondrá el grueso del desarrollo del proyecto.

Como ya se ha comentado, el TFG está compuesto por dos placas principales, que son Arduino y SmartEverything, ambas se programan en lenguaje Arduino (basado en C++), con sus propias librerías y usando el entorno de Arduino IDE. El código en ambas placas será muy distinto, ya que la SmartEverything recibirá información de los sensores de aceleración y GPS y tendrá que llevar a cabo el primer procesamiento de los datos y el Arduino por su parte deberá recibir los datos de la SmartEverything y almacenarlos en la tarjeta de memoria microSD a través del módulo de lectura/escritura.

El esquema principal de funcionamiento, por tanto, quedaría tal que así:

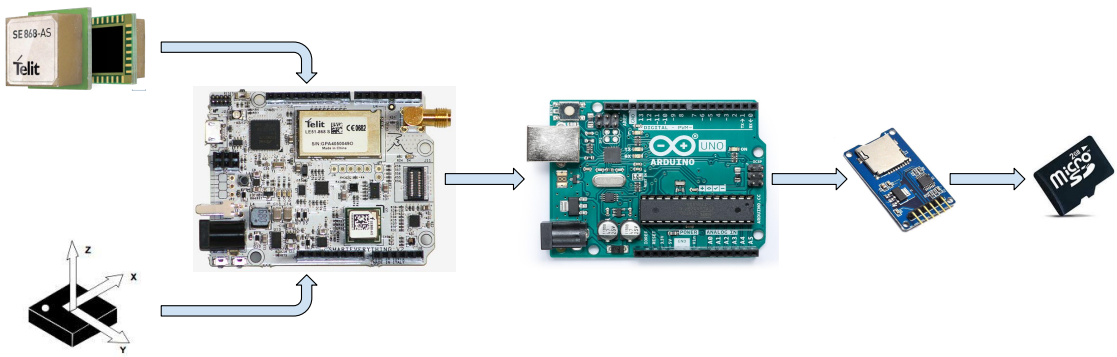

Ilustración 26: Esquema de funcionamiento sistema final

El código completo de ambas placas, así como el análisis de datos final en MATLAB está presente en el **Anexo 1**.

3.3.1 Programación SmartEverything

La programación de la SmartEverything es el elemento principal del TFG, ya que supone la adquisición y procesamiento de los datos. Para ello se ha de apoyar en las librerías propias de los sensores que contiene y en otras suplementarias, tanto para el procesamiento como para el envío de los datos hacia el Arduino.

Para la programación en la placa SmartEverything se deberán tener en cuenta cuatro frentes principales a la hora de programar:

1. Estructura completa del código
2. Uso de librería del módulo GPS SE868-A
3. Uso de librería del módulo acelerómetro LSM9DS1
4. Gestión del envío de datos mediante I^2C

A continuación, se pasa a detallar cómo se han abordado estos cuatro puntos principales y de qué forma se han resuelto los diferentes problemas que se han encontrado durante el desarrollo de los mismos.

3.3.1.1 Estructura completa del código en SmartEverything

El código en la SmartEverything está divido en tres partes principales: `setup`, `loop` y `TC4_Handler`.

- La primera, es la función `void setup()`, en la cuál se realiza primero la inicialización de los módulos Wire, acelerómetro y GPS, así como la inicialización de las variables principales. Tras ello, el grueso de la función setup() recae sobre la inicialización del TC4_Handler, es decir, la interrupción del sistema que se llevará a cabo con una frecuencia de 1Khz.

El código de inicialización de la interrupción, que usa los timer counter 4 y 5 y el reloj genérico 4 es el siguiente:

```
// Set up the generic clock (GCLK4) used to clock timers. 1Khz
REG_GCLK_GENDIV = GCLK_GENDIV_DIV(1) | GCLK_GENDIV_ID(4);
while (GCLK->STATUS.bit.SYNCBUSY);

REG_GCLK_GENCTRL = GCLK_GENCTRL_IDC | GCLK_GENCTRL_GENEN |
                   GCLK_GENCTRL_SRC_DFLL48M | GCLK_GENCTRL_ID(4);
while (GCLK->STATUS.bit.SYNCBUSY);

// Feed GCLK4 to TC4 and TC5
REG_GCLK_CLKCTRL = GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK4 | CLK_CLKCTRL_ID_TC4_TC5;
while (GCLK->STATUS.bit.SYNCBUSY);

REG_TC4_COUNT16_CC0 = 0xEA5F;
while (TC4->COUNT16.STATUS.bit.SYNCBUSY);

NVIC_SetPriority(TC4_IRQn, 0);
NVIC_EnableIRQ(TC4_IRQn);

REG_TC4_INTFLAG |= TC_INTFLAG_OVF;
REG_TC4_INTENSET = TC_INTENSET_OVF;

REG_TC4_CTRLA |= TC_CTRLA_PRESCALER_DIV8 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_ENABLE;
while (TC4->COUNT16.STATUS.bit.SYNCBUSY);
```

Este código provoca que cada 1ms salte la interrupción, la cuál tiene asociada la función `TC4_Handler()` que se explicará más adelante y que supone gran parte de la programación en la SmartEverything, ya que en ella se recogen los valores de latitud y longitud.

- Tras la función `setup()` se encuentra la función `void loop()`, estas dos funciones son comunes en todos los códigos de Arduino y deben estar siempre presentes aunque no realicen ninguna tarea.

La función loop es el núcleo principal del código, en ella se llama a la función `media()`, que será explicada en el uso de la librería GPS, para obtener el valor de latitud y longitud.

```
void loop() {
  if (is_gps_ready == 1) {
    ledRedLight(LOW);
    ledBlueLight(HIGH);
    if (desliza == 1 && medidas_ok == 1) { //Medidas ok evita que se introduzcan ceros de latitud
      ledBlueLight(LOW);                   //y longitud que genera erróneamente el módulo GPS.
      ledGreenLight(HIGH);
      lat_media = media(lat_iter, num_iter); //Media de las últimas num_iter latitudes
      long_media = media(long_iter, num_iter); //Media de las últimas num_iter longitudes
    ...
```

Tras ellos, si no se han cumplido el número de iteraciones iter_loop se almacenan estos datos en sus arrays correspondientes y se obtiene y guarda también el valor de aceleración instantáneo.

```
    ...
    else if (indice < iter_loop) {
        medida.ac[indice] = smeAccelerometer.readZ();
        medida.lat[indice] = lat_media;
        medida.longit[indice] = long_media;
    ...
```

Sin embargo, si se han cumplido el número de iteraciones iter_loop se procede a enviar los datos al Arduino por I^2C.

```
    ...
      if (a == iter_loop) {
        ...
        for (int i = 0; i < 20; i++) {
          Wire.beginTransmission(9);
          ac_save.val = medida.ac[i];
          lat_save.val = medida.lat[i];
          longit_save.val = medida.longit[i];
          Wire.write(ac_save.b, 4);
          Wire.write(lat_save.b, 4);
          Wire.write(longit_save.b, 4);
          Wire.endTransmission();
       ...
```

- Tras el loop el bloque final es `TC4_Handler()`, que contiene todo aquello que se hace cuando ocurre la interrupción. En esencia, este bloque se encarga de comprobar si el módulo GPS está listo y si es el caso toma los valores de latitud y longitud y los introduce deslizando en la variable de la magnitud que le corresponde.

```
if (TC4->COUNT16.INTFLAG.bit.OVF && TC4->COUNT16.INTENSET.bit.OVF)
{
 ...
```

Cuando salta la interrupción se comprueba si se estaban guardando los datos, si no es así (guardando = false), se pasa a recoger los datos de latitud y longitud, comprobando previamente que el GPS esté listo.

```
...
 if (!guardando) {
   if (smeGps.ready()) { //Si el GPS esta listo tomamos datos de ambos sensores
     is_gps_ready = 1;
     latitud   = smeGps.getLatitude();
     longitud  = smeGps.getLongitude();
     if (desliza == 1) { //Si hemos pasado las primeras num_iter iteraciones
       for (int j = num_iter; j > 0; j--) { //Se introducen los valores deslizando
         lat_iter[j] = lat_iter[j - 1];
         long_iter[j] = long_iter[j - 1];
```

```
        }
        lat_iter[0] = latitud;
        long_iter[0] = longitud;
        if (i == num_iter) {i = 0;} else {i++;}
      }
      else { //Si aún no hemos almacenado num_iter valores
        lat_iter[i] = latitud;
        long_iter[i] = longitud;
        if ((lat_iter[i] != 0 && long_iter[i] != 0) || desliza == 1) {
          medidas_ok = 1;
          if (i == num_iter) {
            i = 0;
            desliza = 1;
          }
          else {
            i++;
          }
...
```

Los datos se almacenan deslizando dentro de las variables `lat_iter` y `long_iter`, es decir, cada vez que se recoge un nuevo dato de latitud y longitud este entra a la variable y el primer valor que se había recogido (el que está ahora en la última posición) sale de ella. A estas variables se les hace en la función `loop()` la media y así se está asegurando la precisión y haciendo el error del GPS prácticamente inapreciable.

3.3.1.2 Uso de librería <*sl868a.h*> para módulo GPS SE868-A

La librería para el uso del GPS que contiene la SmartEverything es de uso y distribución libres y está directamente diseñada para facilitar el uso de este módulo en la SmartEverything.

Para el uso de las funciones de esta se ha de incluir la librería mediante `#include <sl868a.h>` lo que hace que se pueda llamar desde el código a todas las funciones disponibles en la misma.

La librería sl868a.h provee de varias funciones a usar en el código:

- `smeGps.begin()`: Con esta función se activa el módulo GPS y comienza la sincronización con los satélites para localizar la posición en la que se encuentra la placa.
- `smeGps.ready()`: Esta función se activa cuando el GPS ha establecido conexión con los satélites y comienza a obtener datos de la posición en la que se encuentra.
- `smeGps.getLatitude()`: Mediante esta función se obtienen los datos de latitud del GPS una vez se ha conectado correctamente con los satélites.
- `smeGps.getLongitude()`: Mediante esta función se obtienen los datos de longitud del GPS una vez se ha conectado correctamente con los satélites.

Gracias a estas funciones se obtienen por tanto los datos de latitud y longitud del GPS. La problemática asociada a esto es que se reciben datos de las mismas demasiado rápido, lo que hace que no sea del todo precisa la posición obtenida. Para solucionar esta problemática se usan 2 listas, una para latitud y otra para longitud, de modo que una vez almacenadas 200 muestras (valor elegido tras varios ensayos) de cada una de las magnitudes se hace la media deslizante de las mismas, esto es, se interpreta como una estructura de datos tipo **FIFO** (First In First Out) en la que se introduce un valor nuevo con cada muestra y se saca de la lista el primero que entró. De este modo se consigue el valor más real posible de la posición, asegurando siempre una alta fiabilidad.

Para hacer el cálculo de la media se desarrolla una función que recibe dos parámetros: el primero de ellos la lista de latitud o longitud y el segundo el número de iteraciones que se van a realizar, el cuál coincide con el número de elementos que compone la lista pasada como primer parámetro. La función hace la suma de los elementos y lo divide por el número total de elementos en la lista, devolviendo así la media:

```
double media(double magnitud[], int num_iter) {
  double suma = 0;
  for (int i = 0; i < num_iter; i++) {
    suma = suma + magnitud[i];
  }
  suma = suma / num_iter;
  return suma;
}
```

La llamada a esta función se lleva a cabo en el bucle principal, mientras que los datos instantáneos de latitud y longitud se obtienen en cada interrupción.

3.3.1.3 Uso de librería <*LSM9DS1.h*> para módulo acelerómetro LSM9DS1

La librería para el uso de acelerómetro que acompaña a la SmartEverything es, como el resto, de uso y distribución libres, supone de gran ayuda a la hora de recibir datos del acelerómetro para su posterior procesamiento.

Para el uso de sus funciones se incluye la librería mediante `#include <LSM9DS1.h>` al igual que se explicó anteriormente en el módulo GPS.

La librería LSM9DS1.h provee de múltiples funciones, solo se usarán dos de ellas (aunque una trendrá tres variantes) en el TFG:

- `smeAccelerometer.begin()`: Mediante esta función se insta al módulo de acelerómetro a comenzar a medir aceleraciones.
- `smeAccelerometer.readZ()`: Mediante esta función se obtiene el dato de aceleración en el eje Z que se está produciendo en la placa en el momento de la llamada. Además de `readZ`, se encuentran `readX` y `readY`, con igual funcionamiento, pero para los ejes X e Y respectivamente. Esta variable habrá de cambiarse convenientemente en el futuro dependiendo de la posición en la que se sitúe la SmartEverything en el tren.

La llamada a la función de activación del módulo se realiza al comienzo del `setup()`, mientras que, la llamada a la función de lectura se lleva a cabo en el bucle principal justo antes de almacenar el valor de la media de latitud y longitud en sus variables correspondientes.

3.3.1.4 Gestión del envío de datos mediante I²C

El protocolo I²C se basa en un bus de datos que interconecta distintas partes de un circuito electrónico y es muy usado para la conexión entre microcontrolador y periféricos. Se basa en una estructura maestro-esclavo, en la que el maestro inicia la comunicación y el esclavo reacciona a este para intercambiar la información necesaria entre ambos.

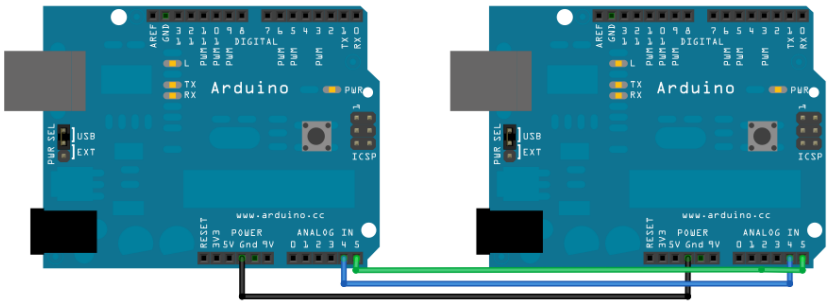

Ilustración 27: Esquemático de dos Arduinos preparados para conexión I²C

Para la comunicación entre Arduinos mediante I²C existe la librería `<Wire.h>`, la cuál establece apoyándose en dos pines analógicos de cada Arduino permite enviar y recibir datos. Para ello utiliza un pin serie de reloj (SCL) con el que el máster generará los pulsos en un intervalo definido y un pin serie de datos (SDA) mediante el que se enviaran los datos entre las dos placas.

Para realizar este envío, como se ha detallado en el punto 3.3.1.1, se utilizan tres funciones de la librería:

- `Wire.beginTransmission(i)`: Indica que va a dar comienzo la transmisión de datos por SDA hacia el dispositivo "i".
- `Wire.write()`: Esta variable es la encargada de mandar los datos por SDA. Recibe dos parámetros, la variable a enviar y el número de bytes que contiene.
- `Wire.endTransmission()`: Indica el fin del envío de datos.

En el caso de la SmartEverything, aunque usando la librería Wire, se usan los puertos TX y RX de conexión serie para la comunicación entre placas. En el Arduino sí que se utilizan los pines analógicos 4 y 5 como figura en el esquemático de la ilustración 26.

3.3.2 Programación Arduino

La programación en el lado del Arduino es más sencilla que en la SmartEverything, ya que en su caso la única función que va a llevar a cabo es recibir los datos provenientes del maestro y guardarlos en la tarjeta SD. Para ello, el código del Arduino se compone de tres partes:

`Setup()`: En el cual se inicializa la conexión I²C mediante `Wire.begin(9)` con ID de dispositivo "9", el mismo al que manda los datos la SmartEverything y tras ello, mediante `Wire.onReceive(receiveEvent)` se avisa al Arduino de que si llegan datos de la SmartEverything se debe gestionar con la función `receiveEvent`. Por último, se inicializa la microSD mediante `SD.begin`, perteneciente a la librería `SD.h`.

```
void setup() {
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  pinMode(LED_BUILTIN, OUTPUT);
  if (!SD.begin(4)) {
    while (1);
  }
}
```

`Loop()`: En el bucle `loop()` se lleva a cabo la escritura en la tarjeta microSD en el fichero "data" mediante las funciones de la librería `SD.h`, que siguen la misma estructura que las funciones `print` y `println` de Arduino.

```
void loop() {
  if(data) {
     File myFile;
     myFile = SD.open("data", FILE_WRITE);
     for(int count = 0; count<60;){
      myFile.print("Datos: ");
      myFile.print(datos[count], 6);
      count++;
...
```

`ReceiveEvent()`: Por último, la función `receiveEvent()` es, como se ha comentado antes, la encargada de leer los datos del bus y gestionarlos. Mediante la función `Wire.read()` se leen los bytes del bus y se introducen en una variable y tras ello en una lista que será posteriormente recorrida en el `loop()`.

```
void receiveEvent(int bytes) {
  while (Wire.available() and !data)
  {
    for (int count = 0; count < 4; count++) {
      x_float.recibido[count] = Wire.read();
    }
    datos[i] = x_float.val;
    i++;
  }
  if(i == 60){
    data=true;
    i=0;
  }
}
```

3.3.3 Análisis de los datos en MATLAB

Una vez obtenidos los datos y almacenados estos en la microSD es necesaria una herramienta que ayude a conocer estos datos y que permita el análisis de los mismos, para ello se decide usar la herramienta de software matemático MATLAB, la cual, mediante un código sencillo como se explicará a continuación permite conocer los resultados obtenidos en el trayecto. El uso de MATLAB está pensado para el análisis de los datos durante el desarrollo del proyecto, con el fin de facilitar la revisión de resultados.

El código en MATLAB se basa en una función que recibe los datos de latitud, longitud y aceleración y lleva a cabo un cálculo de la distancia a partir de latitud y longitud mediante la siguiente función:

```
function distancia=calc_dis(lat1,long1,lat2,long2)
      distancia = (acos(sin(deg2rad(lat1)) * sin(deg2rad(lat2)) +
      cos(deg2rad(lat1)) * cos(deg2rad(lat2)) *cos(deg2rad(long1) -
      deg2rad(long2))) * 6372797.560856);
end
```

La fórmula utilizada permite calcular la distancia entre dos puntos, además gracias a conocer la frecuencia de recepción de los datos (tiempo entre muestras), 1Khz, se puede calcular la velocidad a la que se ha hecho el recorrido, todo esto lo lleva a cabo el bucle for principal del código que es el siguiente:

```
for i=2:tam
     pos(1,i)=mean(lat(max(1,i-n):min(i,tam)));
     pos(2,i)=mean(long(max(1,i-n):min(i,tam)));
     dist(i)=dist(i-1) + calc_dis(lat(i-1),long(i-1),lat(i),long(i));
     dist_media(i)=dist_media(i-1) + calc_dis(pos(1,i-1),pos(2,i-1),
     pos(1,i),pos(2,i));
     if(i>2) %Se han almacenado 2 distancias, se calculan las velocidades
         vel_media(i)=(dist_media(i)-dist_media(i-1))/t_muestra;
         vel(i)=(dist(i)-dist(i-1))/t_muestra;
     end
 end
```

Esta función calcula, mediante las posiciones la distancia que hay entre ambas y tras ello, llevando previamente a cabo el cálculo de la media, calcula la velocidad media y la velocidad instantánea obtenida de las dos muestras de posición.

Estos datos serán representados posteriormente gracias al siguiente código:

```
t=0:t_muestra:(length(lat)*t_muestra-t_muestra);
figure (1)
subplot(2,2,1)
plot(t,vel_media/1000,'r',t,vel/1000,'b')
title('Velocidades')
subplot(2,2,2)
plot(t,dist_media,'r',t,dist,'b')
title('Distancias')
subplot(2,2,3)
plot(dist_media,ac/9.8,'g')
title('Aceleracion')
subplot(2,2,4)
plot3(pos(1,:),pos(2,:),t)
```

Este código dará como resultado gráficas como las siguientes, que aportaran la información necesaria para validar las pruebas que se realicen:

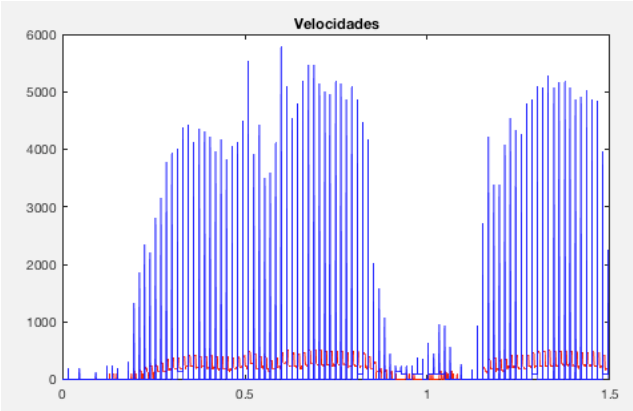

Ilustración 28: Ejemplo de gráfica generada por código MATLAB

Tantos los datos de MATLAB como los almacenados en la SD por el Arduino tienen las mismas unidades, que son las recibidas directamente desde los sensores de la SmartEverything:

- Aceleración: Se mide en mg, que equivale a 10^{-3}g.
- Latitud: Medida en grados decimales.
- Longitud: Medida en grados decimales.

4 PRUEBAS Y EVALUACIÓN

En este apartado se tratarán todas las pruebas llevadas a cabo durante el desarrollo del proyecto, así como las pruebas finales realizadas tras la consecución del objetivo y las conclusiones de estas.

4.1 Pruebas parciales

Durante el desarrollo del TFG se han llevado a cabo pruebas parciales a medida que se iba avanzando en la búsqueda del objetivo final. Las pruebas se dividen en cuatro bloques principales de funcionamiento.

4.1.1 Prueba de librería GPS

Al comienzo del desarrollo de la programación (una vez llevados a cabo los cálculos y la elección de componentes) se realizaron las primeras pruebas del módulo GPS de la SmartEverything. Este módulo supuso numerosos problemas en el inicio del proyecto, ya que la versión más nueva de la librería *sl868a.h* no permitía establecer conexión con el GPS. Esto trajo consigo numerosos problemas, ya que no existía la posibilidad de conocer el problema real que ocurría con la placa, e incluso llevó a intentar probar el código con una placa similar para comprobar que no hubiera ningún error en el módulo de la primera.

El código para la conexión en el inicio era algo simple, ya que solo buscaba conseguir establecer conexión con los satélites y mostrar los datos por pantalla para demostrar que funcionaba correctamente:

```
...
void setup() {
    SerialUSB.begin(115200);
    smeGps.begin();
}
void loop() {
    ...
    if (smeGps.ready()) {

        ledGreenLight(HIGH);
        latitude   = smeGps.getLatitude();
        longitude  = smeGps.getLongitude();

        if ((loop_cnt % 200) == 0) {
            SerialUSB.println(" ");
            SerialUSB.print("Latitud    =  ");
            SerialUSB.println(latitude, 6);
            SerialUSB.print("Longitud   =  ");
            SerialUSB.println(longitude, 6);
        }
    } else {
        if ((loop_cnt % 200) == 0) {
            SerialUSB.println("Locking GPS position...");
            if (led_status == LOW) {
                led_status = HIGH;
            } else {
                led_status = LOW;
            }
    ...
```

Aún siendo un código simple y que, aparentemente, debía funcionar, no era así. La placa no parecía ser capaz de conectarse a los satélites.

Para solucionar este problema y tras mucha investigación al respecto, se decidió pasar a una versión anterior de la librería que gestiona el módulo GPS. Gracias a esto y tras múltiples intentos se consiguió recibir datos reales del módulo:

```
Latitud    =  37.379127
Longitud   =  -6.066661

Latitud    =  37.379131
Longitud   =  -6.066721

Latitud    =  37.380268
Longitud   =  -6.069060
```

Una vez obtenidos los datos, se pasó a confirmar mediante herramientas web que, efectivamente, los datos eran correctos, dando así por finalizadas las primeras pruebas de GPS:

Ilustración 29: Comprobación primeras pruebas GPS

4.1.2 Prueba de la librería del acelerómetro

Tras las pruebas de GPS se pasó a probar la librería del acelerómetro, la cual, a diferencia de la librería del módulo GPS, no dio problemas aparentes. Se creó un pequeño código al estilo del creado para las primeras pruebas del módulo GPS que recogía los valores de aceleración en los tres ejes y los imprimía por puerto serie, esto no ocasionó error alguno por lo que el resultado de la prueba se dio como válido.

El código utilizado fue el siguiente:

```
#include <Arduino.h>
#include <Wire.h>
#include <LSM9DS1.h>

void setup() {
    Wire.begin();
    smeAccelerometer.begin();
    SerialUSB.begin(115200);
}
```

```
void printAxis(int x, int y, int z) {
    SerialUSB.print("  X = ");
    SerialUSB.print(x, DEC);
    SerialUSB.print("     Y = ");
    SerialUSB.print(y, DEC);
    SerialUSB.print("     Z = ");
    SerialUSB.println(z, DEC);
}

void loop() {
    int x = 0;
    int y = 0;
    int z = 0;

    x = smeAccelerometer.readX();
    y = smeAccelerometer.readY();
    z = smeAccelerometer.readZ();

    SerialUSB.print('\n');
    SerialUSB.print("Accelerometer [mg]      :");
    printAxis(x, y, z);

    delay(500);
}
```

Y los resultados de la prueba, que demuestran el movimiento de la placa que se llevó a cabo y por tanto se aceptaron como válidos:

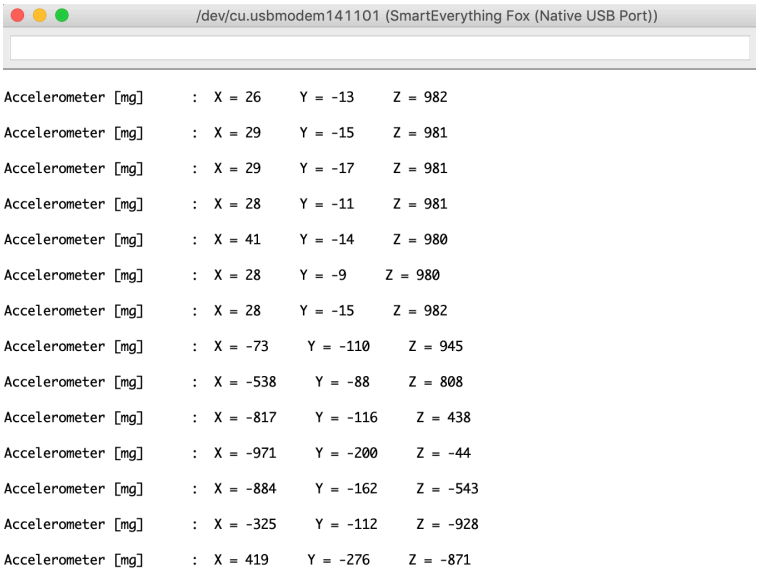

Ilustración 30: Resultados prueba librería acelerómetro

4.1.3 Prueba de comunicación entre las dos placas

Una vez llevadas a cabo y con buenos resultados las pruebas de recepción de los sensores de la SmartEverything se pasó a probar la conectividad entre esta y el Arduino.

Tras numerosas pruebas de conexión sin éxito, se lleva a cabo un estudio para una posible nueva forma de interconexión entre ambas placas, ya que la conexión no se realizaba correctamente mediante los pines A4 y A5 de ambas placas. Tras revisar las posibles conexiones se lleva a cabo un nuevo conexionado usando los pines A4 y A5 del Arduino, como hasta ahora pero cambiando los pines en la SmartEverything al SCL y SDA, que no son más que una réplica de A4 y A5 para conexiones I^2C pero que, sin embargo, sí que permiten realizar la conexión entre ambas placas.

El nuevo esquemático de interconexión quedaría como:

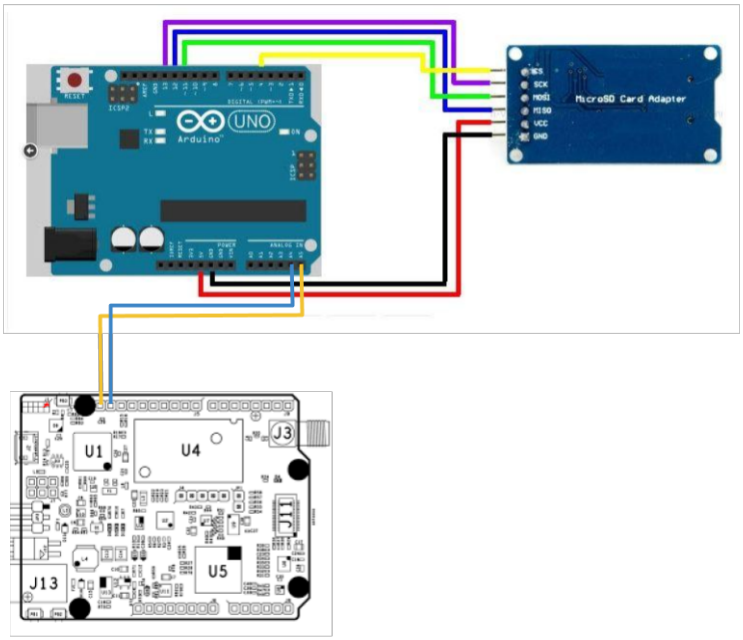

Ilustración 31: Esquemático con modificación SPI

Para ello se usó un pequeño código en cada una de las dos placas, como el siguiente:

- Lado Maestro (SmartEverything):

```
void setup()
{
 Wire.begin();
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(9);
  Wire.write("x is ");
  Wire.write(x);
  Wire.endTransmission();
  x++;
  delay(500);
}
```

- Lado Esclavo (Arduino):

```
void setup()
{
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}
...

void receiveEvent(int howMany)
{
  while(1 < Wire.available())
  {
    char c = Wire.read();
    Serial.print(c);
  }
  int x = Wire.read();
  Serial.println(x);
}
```

Mediante estos dos códigos se lleva a cabo una transmisión desde la SmartEverything al Arduino, el cual tras ello imprime lo recibido por puerto serie. A continuación, el resultado de la ejecución de estos códigos en ambas placas:

Ilustración 32: Resultado prueba comunicación entre las dos placas

Con esta prueba se comprueba que, efectivamente, la transmisión entre ambas placas es correcta y no precisa de una gran complejidad para establecerse.

4.1.4 Prueba de lectura/escritura en microSD

La última prueba parcial que se llevó a cabo fue la escritura en la tarjeta microSD, para ello se utiliza el código de ejemplo de la librería *SD.h* el cual lleva a cabo una escritura y una lectura en la placa, la parte de escritura no se utiliza finalmente en el proyecto, pero era interesante conocer su funcionamiento para futuras posibles mejoras.

```
void setup() {
  ...
  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    while (1);
  }
  ...
  myFile = SD.open("test.txt", FILE_WRITE);

  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");
    myFile.close();
    Serial.println("done.");
  } else {
    Serial.println("error opening test.txt");
  }

  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");
    while (myFile.available()) {
      Serial.write(myFile.read());
    }
    myFile.close();
  } else {
    Serial.println("error opening test.txt");
  }
}
```

Una vez cargado el código en la placa y conectando adecuadamente el Arduino al módulo de lectura/escritura en la microSD se comprueba que el funcionamiento es correcto y que, por tanto, se pueden escribir los datos en la tarjeta de memoria correctamente.

4.2 Pruebas del sistema completo

Una vez concluidas las pruebas parciales se pasa a llevar a cabo las pruebas del sistema completo, lo cual supone el final del proyecto y que trae consigo el análisis de los resultados obtenidos.

La prueba del sistema completo consistió en llevar a cago un recorrido con las placas conectadas a una bicicleta, ya que era el elemento de más fácil acceso al que se podía optar para llevarlas a cabo. Para que las pruebas fuesen totalmente fiables sería ideal realizar pruebas en campo con trenes reales, pero esto supondría grandes costes o la necesidad de un segundo implicado como sería la empresa colaboradora que permitiese dichas pruebas.

De modo que, como se ha comentado, las pruebas se llevaron a cabo con la placa instalada en una bicicleta y haciendo el mismo recorrido varias veces. Recorrido el cual, tenía varias irregularidades conocidas y en el que se hizo una parada para comprobar que, efectivamente, esto se reproducía correctamente en los datos obtenidos de la placa.

El recorrido realizado fue el siguiente:

Ilustración 33: Recorrido de la prueba total

Durante el recorrido hay varios puntos remarcables a tener en cuenta a la hora de analizar los datos, que son:

- 40m: Irregularidad grave en el terreno.
- 60m: Irregularidad leve en el terreno.
- 80m: Irregularidad grave en el terreno.
- 120m: Irregularidad media en el terreno.
- 140m: Irregularidad media en el terreno.
- 180m: Parada en el recorrido y reanudación de la marcha.
- 220m: Irregularidad leve en el terreno.

Una vez realizado el recorrido y recogidos los datos en la tarjeta microSD se pasa a analizarlos mediante MATLAB y su representación gráfica.

En Matlab se sacan 4 gráficas, la primera de ellas representa la velocidad (en azul la velocidad instantánea y en rojo la velocidad media) teniendo en cuenta el tiempo entre muestras. Se puede comprobar que cercano al final hay una bajada de la velocidad a prácticamente cero, que representa la parada en los 180m.

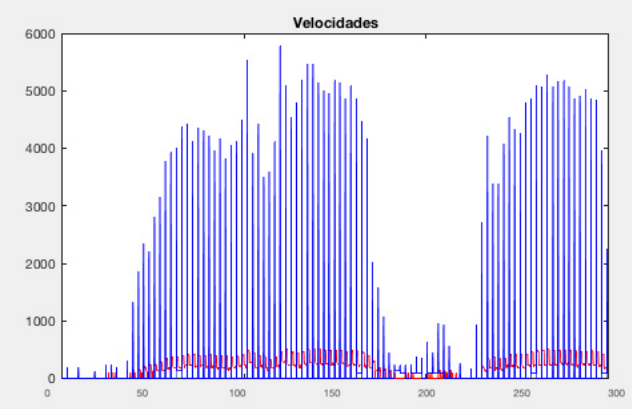

Figura 2: Gráfica de velocidad

La segunda gráfica representa la distancia recorrida frente a las muestras recibidas, en ella se aprecia a la perfección la parada realizada en los casi al final del recorrido (180m).

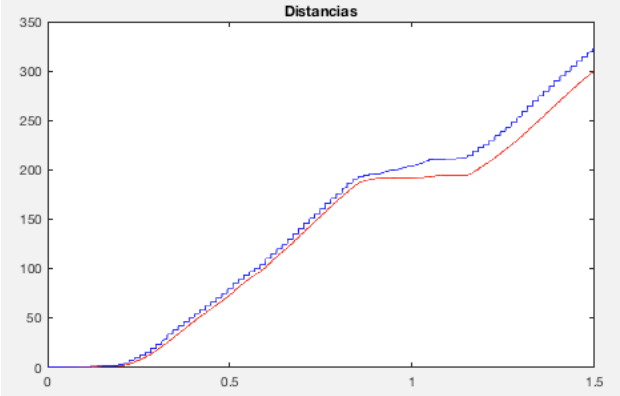

Figura 3: Gráfica de distancia recorrida

La tercera gráfica, quizás la más significativa, representa la aceleración frente a la distancia recorrida. En ella, se aprecian con claridad los picos de aceleración en los puntos indicados previamente (40m, 60m, 80m, etc), lo que prevee un buen funcionamiento del sistema. Además en ella, también se aprecia la parada en torno a los 180m en el recorrido.

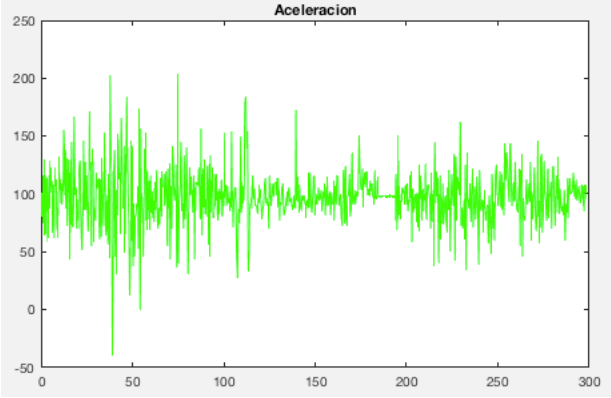

Figura 4: Gráfica de aceleración

Por último, se representa una gráfica orientativa del cambio de latitud y longitud. En ella, lo más relevante es comprobar como el cambio de la latitud es más suave que el de la longitud.

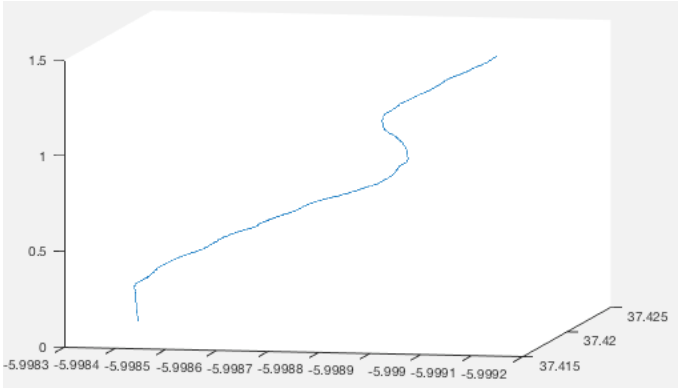

Figura 5: Gráfica de latitud y longitud frente a las muestras recogidas

4.3 Conclusiones de los resultados

Una vez terminadas las pruebas y analizados los datos se llega a una serie de conclusiones:

- El sistema es funcional y detecta, a priori, correctamente las imperfecciones en el terreno.
- El sistema es capaz de controlar las paradas que se puedan realizar en el recorrido del tren.
- Se comprueba que el sistema detecta correctamente los cambios de latitud y longitud.
- El número de muestras y el tiempo entre muestras es correcto, aunque el recorrido que se lleva a cabo es lento y sería necesario probar su eficacia en recorridos de mayor velocidad.
- Para conocer la precisión del sistema a la hora de detectar la gravedad de las irregularidades del terreno, sería necesario realizar una trayectoria en un entorno conocido y controlado.

5 CONCLUSIONES Y DESARROLLOS FUTUROS

Una vez concluido el desarrollo, pruebas y análisis del sistema se pasa a desarrollar las conclusiones y futuras vías de desarrollo que puede seguir el proyecto.

5.1 Consecución de objetivos

Se ha conseguido un sistema autónomo, capaz de detectar las irregularidades en el terreno durante el desarrollo de un trayecto, en este caso en bicicleta, pero extrapolable al de un tren. Todo esto se ha llevado a cabo mediante placas de software libre y de bajo presupuesto, lo que supone una mejora clara al sistema de adquisición de datos de auscultación actual.

El sistema, aun estando en una fase muy primaria de su desarrollo, ya que se podrían realizar múltiples mejoras y pruebas más exhaustivas de él confirma que un desarrollo como este tiene la posibilidad de llevarse a cabo y de una forma no muy compleja, lo que hace prever una posibilidad de cambio en el modo de detectar errores en el carril ferroviario.

Una mejora importante a la hora de confirmar estos resultados, sería llevar a cabo las pruebas en un tren auscultador y llevar a cabo una comparativa con los resultados que genere este. Esto ayudaría a conocer cuan preciso es el sistema y el nivel de mejora que supone frente al sistema actual.

5.2 Desarrollos Futuros

Aún habiendo obtenido resultados satisfactorios de las pruebas realizadas y siendo el sistema funcional, hay numerosas vías por explorar a la hora de continuar con un desarrollo como este. Algunas de estas podrían ser contempladas como desarrollos futuros, como pueden ser:

- Conexión con el puesto de mando (encargado de conocer el estado de los trenes en cada momento y hablar con el maquinista si fuera necesario) en tiempo real. Esto supondría un gran avance, ya que, por ejemplo, si el maquinista nota una irregularidad fuerte durante el trayecto puede avisar al puesto de mando y este, ayudándose de los datos generados por el sistema, puede corroborar si esto es cierto o no para ponerse en contacto de inmediato con el equipo encargado de solucionar una incidencia si así lo fuera.
- Posibilidad de exportarlo a otros campos diferentes al ferrocarril, como pueden ser los automóviles. Exportando el sistema a los automóviles se podría conocer en cada momento el estado real de las carreteras, lo que facilitaría las labores de mantenimiento en este ámbito.
- Información al maquinista en tiempo real. Aunque unida al primer punto, este tiene una ligera diferencia y es que se podría mediante la conexión SIGFOX de la SmartEverything ofrecer datos en tiempo real al maquinista del tren, para dar a conocer a este el estado de la vía y permitirle informar al puesto de mando de una posible irregularidad.
- Inserción de un mayor número de placas en el tren. Esto supondría una mejora en la precisión de los datos adquiridos y, si hubiera un sistema central capaz de comparar los datos de ambas placas y almacenarlos, sería un sistema mucho más completo y fiable.

REFERENCIAS

[1] Patrick L. Walter, «The History of the Accelerometer» Prologue and Epilogue, 2006

[2] «Aviación: Navegando sin perdernos» AviaciónD

[3] Jesús Martínez, Onofre Ricardo Contreras, Susana Aznar, Ángela Lera, «Niveles de actividad física medido con acelerómetro en alumnos de 3º ciclo de Educació Primaria» Revista de Psicología del Deporte 2012. Vol. 21, núm. 1, pp. 117-123

[4] Ruggeduino Board, Rugged Circuits

[5] Omar J. Pérez «Velocidad Relativa entre las placas del Caribe y Suramérica a partir de observaciones GPS en el norte de Venezuela» CONICIT

[6] F. Gómez Bravo «Prototipo para la Monitorización de la Infraestructura Ferroviaria utilizando sensores LIDAR, Imagen de Vídeo y GPS», Universidad de Huelva

[7] Adam Hjort, Måns «Measuring mechanical vibrations using an Arduino as a slave I/O to an EPICS control system», Uppsala University

ANEXO I

Código SmartEverything

```
#include <Arduino.h>
#include <Wire.h>
#include <sl868a.h>
#include <LSM9DS1.h>

int i, primera, desliza, boton, medidas_ok, count, k, indice, b, iter_loop;
boolean guardando;
const int num_iter = 200;
const int num_vel = 20;
const float pi = 3.14159265359;
double lat_iter[num_iter], lat_media, lat_old;
double long_iter[num_iter], long_media, long_old;
double distancia, kilometrico, velocidad[num_vel], v_media;
int ac = 0;
double latitud = 0;
double longitud = 0;
typedef struct {
  int ac[20];
  double lat[20];
  double longit[20];
} save;
save medida;
int is_gps_ready = 0;
union cvt1 { float val; char b[4];} ac_save;
union cvt2 { float val; char b[4];} lat_save;
union cvt3 { float val; char b[4];} longit_save;

// Set timer TC4 to call the TC4_Handler every 10ms
void setup() {
  Wire.begin();
  SerialUSB.begin(115200);
  if (smeAccelerometer.begin() == false) {
    while (1) {
      SerialUSB.print("Accel error");
    }
  }
  smeGps.begin();
  i = 0, k = 0, desliza = 0, primera = 0, kilometrico = 0, medidas_ok = 0,
  boton = 0, indice = 0, b = 0;
  guardando = false;
  iter_loop = 20;
  // Set up the generic clock (GCLK4) used to clock timers. 1Khz
    REG_GCLK_GENDIV = GCLK_GENDIV_DIV(1) |          // Divide the 48MHz clock source by divisor 1:
                                                    48MHz/1=48MHz 1
                      GCLK_GENDIV_ID(4);            // Select Generic Clock (GCLK) 4
  while (GCLK->STATUS.bit.SYNCBUSY);                // Wait for synchronization

  REG_GCLK_GENCTRL = GCLK_GENCTRL_IDC |             // Set the duty cycle to 50/50 HIGH/LOW
                     GCLK_GENCTRL_GENEN |           // Enable GCLK4
                     GCLK_GENCTRL_SRC_DFLL48M |     // Set the 48MHz clock source
                     GCLK_GENCTRL_ID(4);            // Select GCLK4
  while (GCLK->STATUS.bit.SYNCBUSY);                // Wait for synchronization
```

```
  // Feed GCLK4 to TC4 and TC5
  REG_GCLK_CLKCTRL = GCLK_CLKCTRL_CLKEN |         // Enable GCLK4 to TC4 and TC5
                     GCLK_CLKCTRL_GEN_GCLK4 |     // Select GCLK4
                     GCLK_CLKCTRL_ID_TC4_TC5;     // Feed the GCLK4 to TC4 and TC5
  while (GCLK->STATUS.bit.SYNCBUSY);              // Wait for synchronization

   REG_TC4_COUNT16_CC0 = 0xEA5F;                   // Set the TC4 CC0 register as the TOP value in
                                                                          match frequency mode
  while (TC4->COUNT16.STATUS.bit.SYNCBUSY);       // Wait for synchronization

   NVIC_SetPriority(TC4_IRQn, 0);    // Set the Nested Vector Interrupt Controller (NVIC) priority
                                                                        for TC4 to 0 (highest)
  NVIC_EnableIRQ(TC4_IRQn);         // Connect TC4 to Nested Vector Interrupt Controller (NVIC)

  REG_TC4_INTFLAG |= TC_INTFLAG_OVF;              // Clear the interrupt flags
  REG_TC4_INTENSET = TC_INTENSET_OVF;             // Enable TC4 interrupts

  REG_TC4_CTRLA |= TC_CTRLA_PRESCALER_DIV8 |      // Set prescaler to 8, 48MHz/8 = 6MHz
                   TC_CTRLA_WAVEGEN_MFRQ |        // Put the timer TC4 into match frequency (MFRQ)
                                                                                    mode
                   TC_CTRLA_ENABLE;               // Enable TC4
  while (TC4->COUNT16.STATUS.bit.SYNCBUSY);       // Wait for synchronization
}

double media(double magnitud[], int num_iter) {
  double suma = 0;
  for (int i = 0; i < num_iter; i++) {
    suma = suma + magnitud[i];
  }
  suma = suma / num_iter;
  return suma;
}

void loop() {
  if (is_gps_ready == 1) {
    ledRedLight(LOW);
    ledBlueLight(HIGH);
    if (desliza == 1 && medidas_ok == 1) {
      ledBlueLight(LOW);
      ledGreenLight(HIGH);
      lat_media = media(lat_iter, num_iter); //Media de las últimas num_iter latitudes
      long_media = media(long_iter, num_iter); //Media de las últimas num_iter longitudes
      if (indice == iter_loop) {
        guardando = true;
        ledGreenLight(LOW);
        ledBlueLight(HIGH);
        for (int i = 0; i < 20; i++) {
          Wire.beginTransmission(9);
          ac_save.val = medida.ac[i];
          lat_save.val = medida.lat[i];
          longit_save.val = medida.longit[i];
          Wire.write(ac_save.b, 4);
          Wire.write(lat_save.b, 4);
          Wire.write(longit_save.b, 4);
          Wire.endTransmission();
        }
        indice = 0;
      }
```

```
        else if (indice < iter_loop) {
          medida.ac[indice] = smeAccelerometer.readZ();
          medida.lat[indice] = lat_media;
          medida.longit[indice] = long_media;
          indice++;
        }
        guardando = false;
      }
    }
    else {
      SerialUSB.println("Locking GPS position...");
      ledRedLight(HIGH);
    }
  }

  void TC4_Handler()   // Interrupt Service Routine (ISR) for timer TC4
  {
    // Check for overflow (OVF) interrupt
    if (TC4->COUNT16.INTFLAG.bit.OVF && TC4->COUNT16.INTENSET.bit.OVF)
    {
      if (!guardando) {
        if (smeGps.ready()) { //Si el GPS esta listo tomamos datos de ambos sensores.
          is_gps_ready = 1;
          latitud   = smeGps.getLatitude();
          longitud  = smeGps.getLongitude();
          if (desliza == 1) { //Si hemos pasado las primeras num_iter iteraciones
            for (int j = num_iter; j > 0; j--) { //Se introducen los valores deslizando
              lat_iter[j] = lat_iter[j - 1];
              long_iter[j] = long_iter[j - 1];
            }
            lat_iter[0] = latitud;
            long_iter[0] = longitud;
            if (i == num_iter) {
              i = 0;
            }
            else {
              i++;
            }
          }
          else { //Si aún no hemos almacenado num_iter valores
            lat_iter[i] = latitud;
            long_iter[i] = longitud;
            if ((lat_iter[i] != 0 && long_iter[i] != 0) || desliza == 1) {
              medidas_ok = 1;
              if (i == num_iter) {
                i = 0;
                desliza = 1;
              }
              else {
                i++;
              }
            }
          }
        }
        else { //Si el GPS no está listo no tomamos ningún dato
          is_gps_ready = 0;
        }
      }
      REG_TC4_INTFLAG = TC_INTFLAG_OVF;         // Clear the OVF interrupt flag
    }
}
```

Código Arduino

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>

int data=false;
int i = 0;
union cvt {
  float val;
  char recibido[4];
} x_float;
float datos[60];

void setup() {
  Serial.begin(115200);
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  pinMode(LED_BUILTIN, OUTPUT);
  if (!SD.begin(4)) {
    while (1);
  }
}

void loop() {
  if(data) {
     File myFile;
     myFile = SD.open("data", FILE_WRITE);
     for(int count = 0; count<60;){
      myFile.print(datos[count], 6);
      count++;
      myFile.print(" ");
      myFile.print(datos[count], 6);
      count++;
      myFile.print(" ");
      myFile.println(datos[count], 6);
      count++;
     }
     myFile.close();
     data=false;
  }
}

void receiveEvent(int bytes) {
  while (Wire.available() and !data)
  {
    for (int count = 0; count < 4; count++) {
      x_float.recibido[count] = Wire.read();
    }
    datos[i] = x_float.val;
    i++;
  }
  if(i == 60){
    data=true;
    i=0;
  }
}
```

Código MATLAB

```
function tfg(data)
    ac=data(1,:); %Aceleración
    lat=data(2,:); %Latitud
    long=data(3,:); %Longitud
    n=20; %Numero de valores con los que hacemos la media
    vel_media(1:2)=0; %Vector de velocidad media
    vel(1:2)=0; %Vector de velocidad con errores
    dist_media(1)=0; %Vector de distancia media
    dist(1)=0; %Vector de distancia con errores
    pos(1,1)=lat(1:1); %Vector de posición media (lat)
    pos(2,1)=long(1:1); %Vector de posición media (long)
    tam=length(lat);
    t_muestra = 0.001;

    for i=2:tam
        pos(1,i)=mean(lat(max(1,i-n):min(i,tam)));
        pos(2,i)=mean(long(max(1,i-n):min(i,tam)));
        dist(i)=dist(i-1) + calc_dis(lat(i-1),long(i-1),lat(i),long(i));
        dist_media(i)=dist_media(i-1) + calc_dis(pos(1,i-1),pos(2,i-
1),pos(1,i),pos(2,i));
        if(i>2) %Como ya se han almacenado 2 valores de distancia se calculan
las velocidades
            vel_media(i)=(dist_media(i)-dist_media(i-1))/t_muestra;
            vel(i)=(dist(i)-dist(i-1))/t_muestra;
        end
    end
    t=0:t_muestra:(length(lat)*t_muestra-t_muestra);
    figure (1)
    subplot(2,2,1)
    plot(t,vel_media/1000,'r',t,vel/1000,'b')
    title('Velocidades')
    subplot(2,2,2)
    plot(t,dist_media,'r',t,dist,'b')
    title('Distancias')
    subplot(2,2,3)
    plot(dist_media,ac/9.8,'g')
    title('Aceleracion')
    subplot(2,2,4)
    plot3(pos(1,:),pos(2,:),t)
end

function distancia=calc_dis(lat1,long1,lat2,long2) %Función que calcula la
distancia entre dos puntos
    distancia = (acos(sin(deg2rad(lat1)) * sin(deg2rad(lat2))
+cos(deg2rad(lat1)) * cos(deg2rad(lat2)) *cos(deg2rad(long1) -
deg2rad(long2))) * 6372797.560856);
end
```